Plan global de refonte et d'amélioration du projet "Cinephoria"
(en partant de la branche `DEV` et en utilisant uniquement Symfony + Symfony UX/Asset Mapper pour le front)

Contexte rappelé

- Le projet actuel est un monolithe Symfony 7 exposant une API REST (consommée jusqu'ici par du ReactJS),
- L'objectif est de supprimer tout ReactJS/JS lourd au profit de Twig/Stimulus (Symfony UX) et Asset Mapper,
- L'hébergement se fera via GitHub Actions (CI), Docker, et Heroku (ou VPS), en éliminant Alwaysdata,
- Le code doit être clair, maintenable, testable et aisément explicable au jury.

Nous allons parcourir, étape par étape, l'ensemble du projet (entités, contrôleurs, vues, configuration, tests, CI/CD, etc.) pour indiquer ce qu'il faut reprendre, corriger ou implémenter. À chaque paragraphe, vous trouverez :

1. **Constat** – point de départ et problèmes éventuels.
2. **Objectif** – ce qu'il faut atteindre.
3. **Actions détaillées** – instructions précises pour réaliser la refonte ou l'amélioration.

---

1. Préparation de la branche `DEV` et configuration initiale

1.1. Basculer sur la branche DEV

1. Depuis votre copie locale du dépôt (`git clone git@github.com:MDIDKT/RefactoCinephoriaWeb.git`), exécutez :
2. `git fetch origin`
3. `git checkout DEV`
4. Vérifiez que vous êtes bien sur `DEV` et que vous avez pull les derniers commits :
5. `git pull origin DEV`
6. Créez immédiatement une branche de travail dérivée de `DEV`, par exemple :
7. `git checkout -b refacto/symfony-ux`

   Vous travaillerez exclusivement sur `refacto/symfony-ux` jusqu'à la validation finale.

1.2. Mettre à jour `composer.json` et configurer Symfony UX + Asset Mapper

1. **Supprimer les dépendances ReactJS et Node**
   - Ouvrez `composer.json` et retirez toute référence à un "`symfony/webpack-encore-bundle`" ou à des paquets liés à React.
   - Supprimez aussi tout dossier `assets/js/components` ou `react` qui n'est plus utile.
   - Supprimez `package.json` (si présent), le dossier `node_modules/`, et le fichier `webpack.config.js`.

2. **Installer Symfony UX et Asset Mapper**
   - Installez le bundle Symfony UX (Stimulus) ainsi que l'Asset Mapper (introduit dans Symfony 6.4+) :
   - `composer require symfony/ux-twig-component symfony/ux-stimulus symfony/asset`
   - Activez, dans `config/bundles.php`, les lignes correspondantes (elles sont normalement auto-ajoutées) :
   - `return [`
   - `    // ...`
   - `    Symfony\UX\TwigComponent\TwigComponentBundle::class => ['all' => true],`
   - `    Symfony\UX\Stimulus\StimulusBundle::class => ['all' => true],`
   - `    Symfony\Bundle\AssetMapperBundle\AssetMapperBundle::class => ['all' => true],`
   - `];`
3. **Configuration d'Asset Mapper**
   - Créez le dossier `assets/` si ce n'est pas déjà fait, structuré ainsi :
   - `assets/`
   - `├── css/`
   - `│   └── app.css       # votre CSS global (tailwind ou fibre de base)`
   - `├── js/`
   - `│   └── controllers/  # dossier Stimulus controllers`
   - `└── images/           # icônes & images statiques`
   - Dans `config/routes/asset_mapper.yaml`, veillez à avoir :
   - `asset_mapper:`
   - `    resource: "kernel.public_dir/asset_mapper"`
   - `    type: asset_mapper`
   - Dans `twig.yaml`, activez l'extension :
   - `twig:`
   - `    # ...`
   - `    default_path: '%kernel.project_dir%/templates'`
   - `    globals:`
   - `        stimulus_controller_package: '@@stimulus_controller_package'`
4. **Recompiler les assets**
   - Comme nous n'utilisons plus Webpack Encore, il suffit de copier les assets statiques dans `public/` à la main ou via un outil minimaliste :
   - `mkdir -p public/build/css public/build/js public/build/images`
   - `cp -R assets/css/* public/build/css/`
   - `cp -R assets/images/* public/build/images/`
   - Pour les contrôleurs Stimulus, créez un index minimal : `assets/js/controllers/index.js` :
   - `import { Application } from '@hotwired/stimulus';`
   - `import HelloController from './hello_controller'; // exemple`
   -
   - `window.Stimulus = Application.start();`
   - `Stimulus.register('hello', HelloController);`
   - Puis générez un "bundle" minimal :
   - `# si vous voulez compiler via esbuild localement (optionnel)`
   - `npm init -y`
   - `npm install esbuild @hotwired/stimulus`
   - `npx esbuild assets/js/controllers/index.js --bundle --outfile=public/build/js/controllers.js --minify`
   - **Ou** une alternative purement PHP/Asset Mapper : créez un fichier `assets/js/controllers/hello_controller.js` et laissez

Asset Mapper le copier tel quel dans `public/build/js/controllers/hello_controller.js`. Les pages Twig chargeront directement ce script.

1.3. Configuration du local env (.env, .env.local)

1. Vérifiez la connexion à la base de données (PostgreSQL) :
2. `DATABASE_URL="postgresql://db_user:db_pass@127.0.0.1:5432/cinephoria_dev?serverVersion=14&charset=utf8"`
3. Passez la mailer à `null://` en local ou à un service de dev :
4. `MAILER_DSN=null://null`
5. Désactivez eventual `CORS` (si encore présent) ou ajustez la config dans `config/packages/framework.yaml`.

---

2. Revue et refonte des Entités Doctrine

Objectif : s'assurer que chaque entité réponde aux besoins fonctionnels (films, cinémas, salles, séances, réservations, utilisateurs, incidents, avis) avec des relations correctement paramétrées, des validations de champs, et une nomenclature claire.

Fichiers clés à vérifier : `src/Entity/Film.php`, `Cinema.php`, `Room.php` (Salle), `Screening.php` (Séance), `Reservation.php`, `User.php`, `Review.php`, `Incident.php`.

2.1. Entité `User` (Utilisateur)

1. **Constat** :
   - L'entité `User` existante implémente peut-être `UserInterface` + `PasswordAuthenticatedUserInterface`.
   - Vérifiez la présence de champs : `id`, `email`, `roles`, `password`, `firstName`, `lastName`, etc.
   - Peut-être y a-t-il des champs superflus (JWT token) ou manquants (reset password token).
2. **Objectif** :
   - Garder uniquement les champs nécessaires pour la connexion standard via Symfony Security (session + formulaire ou Guard).
   - Gérer les rôles (ROLE_USER, ROLE_EMPLOYEE, ROLE_ADMIN).
   - Prévoir un champ `isVerified` si vous souhaitez confirmation email.
   - Ajouter un champ `resetPasswordToken` (nullable) et `resetPasswordRequestedAt` (DateTime) pour le "mot de passe oublié".
   - Annoter chaque propriété avec les contraintes Symfony Validator (ex : `@Assert\NotBlank`, `@Assert\Email` sur email, etc.).
3. **Actions détaillées** :
   - Ouvrez `src/Entity/User.php`.
   - Vérifiez que la classe ressemble à :
   - `<?php`
   - `namespace App\Entity;`
   -

```
o  use Doctrine\ORM\Mapping as ORM;
o  use Symfony\Component\Security\Core\User\UserInterface;
o  use
   Symfony\Component\Security\Core\User\PasswordAuthenticatedUserI
   nterface;
o  use Symfony\Component\Validator\Constraints as Assert;
o
o  #[ORM\Entity(repositoryClass: UserRepository::class)]
o  class User implements UserInterface,
   PasswordAuthenticatedUserInterface
o  {
o      #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: 'integer')]
o      private ?int $id = null;
o
o      #[ORM\Column(type: 'string', length: 180, unique: true)]
o      #[Assert\NotBlank, Assert\Email]
o      private string $email;
o
o      /** #[ORM\Column(type: 'json')] */
o      #[ORM\Column(type: 'json')]
o      private array $roles = [];
o
o      #[ORM\Column(type: 'string')]
o      private string $password;
o
o      #[ORM\Column(type: 'string', length: 50)]
o      #[Assert\NotBlank]
o      private string $firstName;
o
o      #[ORM\Column(type: 'string', length: 50)]
o      #[Assert\NotBlank]
o      private string $lastName;
o
o      #[ORM\Column(type: 'boolean')]
o      private bool $isVerified = false;
o
o      #[ORM\Column(type: 'string', length: 100, nullable: true)]
o      private ?string $resetPasswordToken = null;
o
o      #[ORM\Column(type: 'datetime', nullable: true)]
o      private ?\DateTimeInterface $resetPasswordRequestedAt =
   null;
o
o      // + getters & setters
o      public function getId(): ?int { return $this->id; }
o      public function getEmail(): string { return $this->email; }
o      public function setEmail(string $email): self { $this-
   >email = $email; return $this; }
o      public function getUserIdentifier(): string { return $this-
   >email; }
o      public function getRoles(): array { return
   array_unique(array_merge($this->roles, ['ROLE_USER'])); }
o      public function setRoles(array $roles): self { $this->roles
   = $roles; return $this; }
o      public function getPassword(): string { return $this-
   >password; }
o      public function setPassword(string $password): self {
   $this->password = $password; return $this; }
o      public function eraseCredentials() { /* vider champs
   temporaires si besoin */ }
o      // FirstName / LastName / isVerified / resetPasswordToken …
```

- `}`
- Supprimez tout champ lié à JWT (si vous prévoyez d'abandonner JWT au profit de la sécurité session/forms).
- Mettez à jour la repository `UserRepository.php` au besoin (par exemple, méthode pour retrouver un utilisateur via `resetPasswordToken`).
- **Générez une migration** :
- `php bin/console make:migration`
- `php bin/console doctrine:migrations:migrate`

2.2. Entité `Cinema` (Cinéma)

1. **Constat** :
   - Probablement existante avec des champs comme `id`, `name`, `address`, `city`, `postalCode`, `phoneNumber`.
   - Vérifier si les relations vers `Room` sont bidirectionnelles ou non, etc.
2. **Objectif** :
   - S'assurer que chaque cinéma possède plusieurs salles (`Room`), et que les séances (`Screening`) sont liées à un `Room`.
   - Ajouter un champ `isActive` (bool) si vous souhaitez activer ou désactiver un cinéma.
   - Valider la cohérence des données (ex : nom non vide, adresse non vide).
3. **Actions détaillées** :
   - Ouvrez `src/Entity/Cinema.php` :

```
<?php
namespace App\Entity;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: CinemaRepository::class)]
class Cinema
{
    #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: 'integer')]
    private ?int $id = null;

    #[ORM\Column(type: 'string', length: 100)]
    #[Assert\NotBlank]
    private string $name;

    #[ORM\Column(type: 'string', length: 255)]
    #[Assert\NotBlank]
    private string $address;

    #[ORM\Column(type: 'string', length: 100)]
    #[Assert\NotBlank]
    private string $city;

    #[ORM\Column(type: 'string', length: 10)]
    #[Assert\NotBlank]
    private string $postalCode;

    #[ORM\Column(type: 'string', length: 20)]
    #[Assert\NotBlank]
    private string $phoneNumber;
```

```
    #[ORM\Column(type: 'boolean')]
    private bool $isActive = true;

    #[ORM\OneToMany(mappedBy: 'cinema', targetEntity:
Room::class, cascade: ['persist', 'remove'])]
    private Collection $rooms;

    public function __construct()
    {
        $this->rooms = new ArrayCollection();
    }

    // getters & setters ...
    public function getId(): ?int { return $this->id; }
    public function getName(): string { return $this->name; }
    public function setName(string $name): self { $this->name =
$name; return $this; }
    // address, city, postalCode, phoneNumber, isActive ...

    /** @return Collection|Room[] */
    public function getRooms(): Collection { return $this-
>rooms; }

    public function addRoom(Room $room): self
    {
        if (!$this->rooms->contains($room)) {
            $this->rooms[] = $room;
            $room->setCinema($this);
        }
        return $this;
    }

    public function removeRoom(Room $room): self
    {
        if ($this->rooms->removeElement($room)) {
            if ($room->getCinema() === $this) {
                $room->setCinema(null);
            }
        }
        return $this;
    }
}
```

- Vérifiez la relation dans `Room.php` (voir ci-dessous).
- Générez migration si changement.

2.3. Entité `Room` (Salle)

1. **Constat** :
   - Définie pour représenter une salle physique d'un cinéma, champs attendus : `id`, `name` ou `number`, `capacity`(nombre de sièges), `accessibleSeats` (nombre de places PMR), `projectionQuality` (ex : "4DX", "3D", "4K"), relation ManyToOne vers `Cinema`.
2. **Objectif** :
   - Assurer que chaque salle appartient exactement à un cinéma (relation non nul).
   - Valider `capacity` et `accessibleSeats` (positif, accessibleSeats ≤ capacity).
   - Prévoir une relation OneToMany vers `Screening` (séances) pour faciliter la récupération des séances par salle.

3. **Actions détaillées** :
   - Ouvrez `src/Entity/Room.php` :

```
<?php
namespace App\Entity;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: RoomRepository::class)]
class Room
{
    #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: 'integer')]
    private ?int $id = null;

    #[ORM\Column(type: 'string', length: 20)]
    #[Assert\NotBlank]
    private string $number; // ou “name”

    #[ORM\Column(type: 'integer')]
    #[Assert\Positive]
    private int $capacity;

    #[ORM\Column(type: 'integer')]
    #[Assert\PositiveOrZero]
    private int $accessibleSeats = 0;

    #[ORM\Column(type: 'string', length: 50)]
    #[Assert\NotBlank]
    private string $projectionQuality;

    #[ORM\ManyToOne(targetEntity: Cinema::class, inversedBy:
'rooms')]
    #[ORM\JoinColumn(nullable: false)]
    private Cinema $cinema;

    #[ORM\OneToMany(mappedBy: 'room', targetEntity:
Screening::class, cascade: ['persist', 'remove'])]
    private Collection $screenings;

    public function __construct()
    {
        $this->screenings = new ArrayCollection();
    }

    // getters & setters...
    public function getId(): ?int { return $this->id; }
    public function getNumber(): string { return $this->number;
}
    public function setNumber(string $number): self { $this-
>number = $number; return $this; }
    public function getCapacity(): int { return $this-
>capacity; }
    public function setCapacity(int $capacity): self { $this-
>capacity = $capacity; return $this; }
    public function getAccessibleSeats(): int { return $this-
>accessibleSeats; }
    public function setAccessibleSeats(int $accessibleSeats):
self { $this->accessibleSeats = $accessibleSeats; return $this;
}
```

```
    public function getProjectionQuality(): string { return $this->projectionQuality; }
    public function setProjectionQuality(string $quality): self { $this->projectionQuality = $quality; return $this; }
    public function getCinema(): Cinema { return $this->cinema; }
    public function setCinema(Cinema $cinema): self { $this->cinema = $cinema; return $this; }

    /** @return Collection|Screening[] */
    public function getScreenings(): Collection { return $this->screenings; }
    // addScreening(), removeScreening() similaires à Cinema<->Room
}
```

- Générez une migration si besoin.

2.4. Entité `Film`

1. **Constat** :
   - Devrait contenir au minimum : `id`, `title`, `description`, `ageRating` (âge minimum), `posterPath` (lien vers affiche), `releaseDate`, `isFeatured` (coup de cœur), `averageRating` (moyenne calculée des avis), relation ManyToMany vers `Cinema` (si vous stockez « dans quel cinéma ce film est projeté »).
2. **Objectif** :
   - Assurer que chaque film peut être projeté dans plusieurs cinémas, et qu'on puisse filtrer par cinéma facilement.
   - Un champ `createdAt` (DateTimeImmutable) pour savoir quel jour a été ajouté le film (utile pour "dernier mercredi").
   - Les champs sont tous obligatoires à l'exception du `posterPath` (nullable si absent).
   - Prévoir une relation OneToMany vers `Review` (avis) pour calculer la note moyenne.
3. **Actions détaillées** :
   - Ouvrez `src/Entity/Film.php` :

```
<?php
namespace App\Entity;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: FilmRepository::class)]
class Film
{
    #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: 'integer')]
    private ?int $id = null;

    #[ORM\Column(type: 'string', length: 150)]
    #[Assert\NotBlank]
    private string $title;

    #[ORM\Column(type: 'text')]
    #[Assert\NotBlank]
    private string $description;
```

```
o
o       #[ORM\Column(type: 'integer')]
o       #[Assert\NotBlank, Assert\Range(min: 0, max: 18)]
o       private int $ageRating;
o
o       #[ORM\Column(type: 'string', length: 255, nullable: true)]
o       private ?string $posterPath = null;
o
o       #[ORM\Column(type: 'datetime')]
o       private \DateTimeInterface $releaseDate;
o
o       #[ORM\Column(type: 'boolean')]
o       private bool $isFeatured = false;
o
o       #[ORM\Column(type: 'float')]
o       private float $averageRating = 0.0;
o
o       #[ORM\Column(type: 'datetime')]
o       private \DateTimeInterface $createdAt;
o
o       #[ORM\ManyToMany(targetEntity: Cinema::class, inversedBy:
    'films')]
o       private Collection $cinemas;
o
o       #[ORM\OneToMany(mappedBy: 'film', targetEntity:
    Screening::class, cascade: ['persist', 'remove'])]
o       private Collection $screenings;
o
o       #[ORM\OneToMany(mappedBy: 'film', targetEntity:
    Review::class, cascade: ['persist', 'remove'])]
o       private Collection $reviews;
o
o       public function __construct()
o       {
o           $this->cinemas    = new ArrayCollection();
o           $this->screenings = new ArrayCollection();
o           $this->reviews    = new ArrayCollection();
o           $this->createdAt  = new \DateTimeImmutable();
o       }
o
o       // getters & setters ...
o       public function getId(): ?int { return $this->id; }
o       public function getTitle(): string { return $this->title; }
o       public function setTitle(string $title): self { $this-
    >title = $title; return $this; }
o       public function getDescription(): string { return $this-
    >description; }
o       public function setDescription(string $desc): self { $this-
    >description = $desc; return $this; }
o       public function getAgeRating(): int { return $this-
    >ageRating; }
o       public function setAgeRating(int $rate): self { $this-
    >ageRating = $rate; return $this; }
o       public function getPosterPath(): ?string { return $this-
    >posterPath; }
o       public function setPosterPath(?string $path): self { $this-
    >posterPath = $path; return $this; }
o       public function getReleaseDate(): \DateTimeInterface {
    return $this->releaseDate; }
o       public function setReleaseDate(\DateTimeInterface $date):
    self { $this->releaseDate = $date; return $this; }
```

```
    public function getIsFeatured(): bool { return $this->isFeatured; }
    public function setIsFeatured(bool $feat): self { $this->isFeatured = $feat; return $this; }
    public function getAverageRating(): float { return $this->averageRating; }
    public function setAverageRating(float $avg): self { $this->averageRating = $avg; return $this; }
    public function getCreatedAt(): \DateTimeInterface { return $this->createdAt; }
    public function setCreatedAt(\DateTimeInterface $createdAt): self { $this->createdAt = $createdAt; return $this; }

    /** @return Collection|Cinema[] */
    public function getCinemas(): Collection { return $this->cinemas; }
    public function addCinema(Cinema $cinema): self {
        if (!$this->cinemas->contains($cinema)) {
            $this->cinemas->add($cinema);
        }
        return $this;
    }
    public function removeCinema(Cinema $cinema): self {
        $this->cinemas->removeElement($cinema);
        return $this;
    }

    /** @return Collection|Screening[] */
    public function getScreenings(): Collection { return $this->screenings; }
    // addScreening(), removeScreening() ...

    /** @return Collection|Review[] */
    public function getReviews(): Collection { return $this->reviews; }
    // addReview(), removeReview() ...

    // Méthode pour recalculer la note moyenne
    public function recalcAverageRating(): self
    {
        $count = count($this->reviews);
        if ($count === 0) {
            $this->averageRating = 0.0;
        } else {
            $sum = array_reduce($this->reviews->toArray(), fn($carry, Review $r) => $carry + $r->getRating(), 0);
            $this->averageRating = round($sum / $count, 2);
        }
        return $this;
    }
}
```

- N’oubliez pas d’ajouter dans `Cinema.php` l’attribut inverse :

```
#[ORM\ManyToMany(targetEntity: Film::class, mappedBy: 'cinemas')]
private Collection $films;
```

- Générez une migration si modifié.

2.5. Entité `Screening` (Séance)

1. **Constat** :
    - Elle modélise une séance précise, champs attendus : `id`, `startTime`, `endTime`, relation ManyToOne vers `Room`, ManyToOne vers `Film`, prix (par qualité), placesDisponibles (calculé dynamiquement ou stocké à chaque nouvel état).
2. **Objectif** :
    - Implémenter un champ `placesAvailable` qui se met à jour à chaque réservation.
    - Vérifier que `startTime < endTime`.
    - Ajouter `price4dx`, `price3d`, `price4k`, etc. **Ou** simplifier en `prices` JSON (clé = qualité, valeur = montant).
    - Relation OneToMany vers `Reservation` pour le lien direct.
3. **Actions détaillées** :
    - Ouvrez `src/Entity/Screening.php` :
    - ```
      <?php
      namespace App\Entity;

      use Doctrine\Common\Collections\ArrayCollection;
      use Doctrine\Common\Collections\Collection;
      use Doctrine\ORM\Mapping as ORM;
      use Symfony\Component\Validator\Constraints as Assert;

      #[ORM\Entity(repositoryClass: ScreeningRepository::class)]
      class Screening
      {
          #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: 'integer')]
          private ?int $id = null;

          #[ORM\Column(type: 'datetime')]
          #[Assert\NotBlank]
          private \DateTimeInterface $startTime;

          #[ORM\Column(type: 'datetime')]
          #[Assert\NotBlank, Assert\Expression("this.getEndTime() > this.getStartTime()", message: "L'heure de fin doit être postérieure à l'heure de début.")]
          private \DateTimeInterface $endTime;

          #[ORM\Column(type: 'json')]
          #[Assert\NotBlank]
          private array $prices = []; // ex: ['4DX'=>15.00, '3D'=>12.00, '4K'=>10.00]

          #[ORM\Column(type: 'integer')]
          #[Assert\PositiveOrZero]
          private int $placesAvailable;

          #[ORM\ManyToOne(targetEntity: Film::class, inversedBy: 'screenings')]
          #[ORM\JoinColumn(nullable: false)]
          private Film $film;

          #[ORM\ManyToOne(targetEntity: Room::class, inversedBy: 'screenings')]
          #[ORM\JoinColumn(nullable: false)]
          private Room $room;

          #[ORM\OneToMany(mappedBy: 'screening', targetEntity: Reservation::class, cascade: ['persist', 'remove'])]
      ```
      ```

```
o      private Collection $reservations;
o
o      public function __construct()
o      {
o          $this->reservations = new ArrayCollection();
o          // placesAvailable sera initialisé après injection de
   la salle (room)
o      }
o
o      // getters & setters...
o      public function getId(): ?int { return $this->id; }
o      public function getStartTime(): \DateTimeInterface { return
   $this->startTime; }
o      public function setStartTime(\DateTimeInterface $time):
   self { $this->startTime = $time; return $this; }
o      public function getEndTime(): \DateTimeInterface { return
   $this->endTime; }
o      public function setEndTime(\DateTimeInterface $time): self
   { $this->endTime = $time; return $this; }
o      public function getPrices(): array { return $this->prices;
   }
o      public function setPrices(array $prices): self { $this-
   >prices = $prices; return $this; }
o      public function getPlacesAvailable(): int { return $this-
   >placesAvailable; }
o      public function setPlacesAvailable(int $places): self {
   $this->placesAvailable = $places; return $this; }
o      public function getFilm(): Film { return $this->film; }
o      public function setFilm(Film $film): self { $this->film =
   $film; return $this; }
o      public function getRoom(): Room { return $this->room; }
o      public function setRoom(Room $room): self {
o          $this->room = $room;
o          // initialiser placesAvailable si c'est le premier set.
o          if ($this->placesAvailable === 0) {
o              $this->placesAvailable = $room->getCapacity();
o          }
o          return $this;
o      }
o
o      /** @return Collection|Reservation[] */
o      public function getReservations(): Collection { return
   $this->reservations; }
o      public function addReservation(Reservation $res): self {
o          if (!$this->reservations->contains($res)) {
o              $this->reservations->add($res);
o              $res->setScreening($this);
o              $this->placesAvailable -= $res->getSeatsBooked();
   // décrémente
o          }
o          return $this;
o      }
o      public function removeReservation(Reservation $res): self {
o          if ($this->reservations->removeElement($res)) {
o              if ($res->getScreening() === $this) {
o                  $res->setScreening(null);
o                  $this->placesAvailable += $res-
   >getSeatsBooked(); // remet
o              }
o          }
o          return $this;
```

- `    }`
- `}`
- **Remarque** : ajustez la logique de décrémentation/remontée des places en fonction du champ `seatsBooked`de `Reservation`.
- Générez la migration si modifié.

2.6. Entité `Reservation`

1. **Constat** :
   - Elle lie un `User` à une `Screening`, champs attendus : `id`, `createdAt`, `seatsBooked` (nombre de places réservées), `seatNumbers` (tableau de numéros de sièges choisis), `totalPrice`, `status` (ex : “PENDING”, “CONFIRMED”, “CANCELLED”), `qrCodeToken` (pour QR code), relation ManyToOne vers `User` et `Screening`.
2. **Objectif** :
   - Valider que `seatsBooked` ≤ `screening.placesAvailable` avant persistance.
   - Générer un jeton aléatoire `qrCodeToken` (string unique) à la création pour le QR code.
   - Calculer `totalPrice` automatiquement (prix unitaire via `screening.prices[qualité] * seatsBooked`).
   - S’assurer que `seatNumbers` (JSON) correspond à un tableau d’entiers (numéros de sièges).
3. **Actions détaillées** :
   - Ouvrez `src/Entity/Reservation.php` :

```
<?php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: ReservationRepository::class)]
class Reservation
{
    #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: 'integer')]
    private ?int $id = null;

    #[ORM\Column(type: 'datetime')]
    private \DateTimeInterface $createdAt;

    #[ORM\Column(type: 'integer')]
    #[Assert\Positive]
    private int $seatsBooked;

    #[ORM\Column(type: 'json')]
    #[Assert\NotBlank]
    private array $seatNumbers = []; // ex: [1,2,3]

    #[ORM\Column(type: 'float')]
    private float $totalPrice;

    #[ORM\Column(type: 'string', length: 100, unique: true)]
    private string $qrCodeToken;

    #[ORM\Column(type: 'string', length: 20)]
```

```
    #[Assert\Choice(choices: ['PENDING', 'CONFIRMED',
'CANCELLED'])]
    private string $status = 'PENDING';

    #[ORM\ManyToOne(targetEntity: User::class, inversedBy:
'reservations')]
    #[ORM\JoinColumn(nullable: false)]
    private User $user;

    #[ORM\ManyToOne(targetEntity: Screening::class, inversedBy:
'reservations')]
    #[ORM\JoinColumn(nullable: false)]
    private Screening $screening;

    public function __construct()
    {
        $this->createdAt = new \DateTimeImmutable();
        $this->qrCodeToken = bin2hex(random_bytes(16)); // ex:
32 chars hex
    }

    // getters & setters...
    public function getId(): ?int { return $this->id; }
    public function getCreatedAt(): \DateTimeInterface { return
$this->createdAt; }
    public function getSeatsBooked(): int { return $this-
>seatsBooked; }
    public function setSeatsBooked(int $count): self {
        $this->seatsBooked = $count;
        return $this;
    }
    public function getSeatNumbers(): array { return $this-
>seatNumbers; }
    public function setSeatNumbers(array $nums): self {
        $this->seatNumbers = $nums;
        return $this;
    }
    public function getTotalPrice(): float { return $this-
>totalPrice; }
    public function setTotalPrice(float $price): self { $this-
>totalPrice = $price; return $this; }
    public function getQrCodeToken(): string { return $this-
>qrCodeToken; }
    // pas de setter pour qrCodeToken, on génère en __construct

    public function getStatus(): string { return $this->status;
}
    public function setStatus(string $status): self { $this-
>status = $status; return $this; }

    public function getUser(): User { return $this->user; }
    public function setUser(User $user): self { $this->user =
$user; return $this; }
    public function getScreening(): Screening { return $this-
>screening; }
    public function setScreening(Screening $screening): self {
$this->screening = $screening; return $this; }
}
```

- Dans `Screening::addReservation()`, vous avez déjà la logique pour décrémenter les places.
- Dans `Screening::removeReservation()`, la logique inverse.

- **Important** : dans le contrôleur de réservation, avant d'appeler `$entityManager->persist($reservation)`, vérifiez `if ($screening->getPlacesAvailable() < $seatsBooked)` et refusez la requête.
- Générez migration si modifié.

2.7. Entité `Review` (Avis / Note)

1. **Constat** :
   - Probablement implémentée avec champs : `id`, `rating` (int de 1 à 5), `comment` (texte optionnel), `createdAt`, relation ManyToOne vers `User`, relation ManyToOne vers `Film`, `status` (PENDING/VALIDATED).
2. **Objectif** :
   - Valider que `rating` ∈ [1,5] et que chaque utilisateur ne peut noter un même film qu'une seule fois.
   - Mettre par défaut `status = 'PENDING'` (en attente de validation par un employé), et proposer plusieurs méthodes dans le repository pour récupérer uniquement les avis validés (`status = 'VALIDATED'`).
   - Ajouter une contrainte d'unicité sur (`user_id`, `film_id`) pour empêcher le multi-avis.
3. **Actions détaillées** :
   - Ouvrez `src/Entity/Review.php` :

```
<?php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: ReviewRepository::class)]
#[ORM\UniqueConstraint(name: "one_review_per_user_per_film", columns: ["user_id", "film_id"])]
class Review
{
    #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: 'integer')]
    private ?int $id = null;

    #[ORM\Column(type: 'integer')]
    #[Assert\Range(min:1, max:5)]
    private int $rating;

    #[ORM\Column(type: 'text', nullable: true)]
    private ?string $comment = null;

    #[ORM\Column(type: 'datetime')]
    private \DateTimeInterface $createdAt;

    #[ORM\Column(type: 'string', length: 20)]
    #[Assert\Choice(choices: ['PENDING', 'VALIDATED', 'REJECTED'])]
    private string $status = 'PENDING';

    #[ORM\ManyToOne(targetEntity: User::class, inversedBy: 'reviews')]
    #[ORM\JoinColumn(nullable: false)]
    private User $user;

```

```
    #[ORM\ManyToOne(targetEntity: Film::class, inversedBy:
'reviews')]
    #[ORM\JoinColumn(nullable: false)]
    private Film $film;

    public function __construct()
    {
        $this->createdAt = new \DateTimeImmutable();
    }

    // getters & setters...
    public function getId(): ?int { return $this->id; }
    public function getRating(): int { return $this->rating; }
    public function setRating(int $rating): self { $this-
>rating = $rating; return $this; }
    public function getComment(): ?string { return $this-
>comment; }
    public function setComment(?string $comment): self { $this-
>comment = $comment; return $this; }
    public function getCreatedAt(): \DateTimeInterface { return
$this->createdAt; }
    public function getStatus(): string { return $this->status;
}
    public function setStatus(string $status): self { $this-
>status = $status; return $this; }
    public function getUser(): User { return $this->user; }
    public function setUser(User $user): self { $this->user =
$user; return $this; }
    public function getFilm(): Film { return $this->film; }
    public function setFilm(Film $film): self { $this->film =
$film; return $this; }
}
```

- Dans `Film::recalcAverageRating()`, vous recalculerez la moyenne uniquement sur les reviews validés (filtrage à faire au moment de l'agrégation).
- Générez migration.

2.8. Entité `Incident` (Incident d'une séance)

1. **Constat** :
   - Destinée aux employés pour signaler un problème dans une salle pendant une séance. Champs attendus : `id`, `description`, `createdAt`, relation ManyToOne vers `Room` (ou `Screening`), relation ManyToOne vers `User` (employé).
2. **Objectif** :
   - Stocker la nature de l'incident, la date, l'employé à l'origine, et si besoin un statut (résolu/ non résolu).
3. **Actions détaillées** :
   - Créez/éditez `src/Entity/Incident.php` :

```
<?php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: IncidentRepository::class)]
class Incident
{
```

```
    #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: 'integer')]
    private ?int $id = null;

    #[ORM\Column(type: 'text')]
    #[Assert\NotBlank]
    private string $description;

    #[ORM\Column(type: 'datetime')]
    private \DateTimeInterface $createdAt;

    #[ORM\Column(type: 'string', length: 20)]
    #[Assert\Choice(choices: ['OPEN', 'IN_PROGRESS',
'RESOLVED', 'CLOSED'])]
    private string $status = 'OPEN';

    #[ORM\ManyToOne(targetEntity: Room::class)]
    #[ORM\JoinColumn(nullable: false)]
    private Room $room;

    #[ORM\ManyToOne(targetEntity: User::class)]
    #[ORM\JoinColumn(nullable: false)]
    private User $reporter; // l'employé qui a signalé

    public function __construct()
    {
        $this->createdAt = new \DateTimeImmutable();
    }

    // getters & setters...
    public function getId(): ?int { return $this->id; }
    public function getDescription(): string { return $this-
>description; }
    public function setDescription(string $desc): self { $this-
>description = $desc; return $this; }
    public function getCreatedAt(): \DateTimeInterface { return
$this->createdAt; }
    public function getStatus(): string { return $this->status;
}
    public function setStatus(string $status): self { $this-
>status = $status; return $this; }
    public function getRoom(): Room { return $this->room; }
    public function setRoom(Room $room): self { $this->room =
$room; return $this; }
    public function getReporter(): User { return $this-
>reporter; }
    public function setReporter(User $u): self { $this-
>reporter = $u; return $this; }
}
```

- Générez migration.

---

3. Revue et refonte des Repositories / Requêtes personnalisées

Chaque entité a son repository. Pour garantir que le code reste maintenable, placez toute logique de requête complexe dans le repository adéquat. Par exemple :

3.1. `FilmRepository`

- **Constat** : probablement existant, mais peut manquer de requêtes pour récupérer :
  1. Les « films ajoutés le dernier mercredi » pour la page d'accueil,
  2. Les films filtrés par cinéma, genre ou jour,
  3. Les films "coup de cœur" (isFeatured = true).
- **Objectif** :
  - Implémenter des méthodes claires :
  - `public function findLastWednesdayFilms(): array;`
  - `public function findByCinemaGenreDay(int $cinemaId = null, string $genre = null, \DateTimeInterface $day = null): QueryBuilder;`
  - `public function findFeatured(): array;`
- **Actions détaillées** :
  1. Ouvrez `src/Repository/FilmRepository.php` et ajoutez :

```
public function findLastWednesdayFilms(): array
{
    // Trouver le dernier mercredi
    $now = new \DateTimeImmutable();
    $lastWednesday = $now->modify('last Wednesday');
    // On cherche tous les films dont createdAt (mercredi) correspond au jour du lastWednesday
    return $this->createQueryBuilder('f')
        ->andWhere('DATE(f.createdAt) = :wed')
            ->setParameter('wed', $lastWednesday->format('Y-m-d'))
            ->orderBy('f.releaseDate', 'DESC')
            ->getQuery()
            ->getResult();
   }

   public function findByCinemaGenreDay($cinemaId = null, $genre = null, \DateTimeInterface $day = null)
   {
       $qb = $this->createQueryBuilder('f')
           ->leftJoin('f.cinemas', 'c')
           ->leftJoin('f.screenings', 's')
           ->addSelect('c', 's');

       if ($cinemaId) {
           $qb->andWhere('c.id = :cinema')->setParameter('cinema', $cinemaId);
       }
       if ($genre) {
           $qb->andWhere('f.genre = :genre')->setParameter('genre', $genre);
           // Si vous stockez plusieurs genres en JSON, adapter la condition
       }
       if ($day) {
           $qb->andWhere('DATE(s.startTime) = :day')->setParameter('day', $day->format('Y-m-d'));
       }

       return $qb->orderBy('f.title', 'ASC')->getQuery()->getResult();
   }

   public function findFeatured(): array
```

```
    {
        return $this->createQueryBuilder('f')
            ->andWhere('f.isFeatured = :feat')->setParameter('feat', true)
            ->orderBy('f.releaseDate','DESC')
            ->getQuery()
            ->getResult();
    }
```

45. Pour la pagination éventuelle, vous pouvez renvoyer un `QueryBuilder` et le paginer dans le contrôleur.

3.2. `ScreeningRepository`

- **Constat** : doit fournir les séances d'un cinéma/un film sur un jour donné avec la disponibilité.
- **Objectif** :
  - Requête pour récupérer toutes les séances (avec film et salle) afin d'afficher une grille par jour/cinéma,
  - Méthode pour vérifier la disponibilité avant l'affichage (ex : placesAvailable > 0).
- **Actions détaillées** :

```
public function findAvailableByCinemaAndDay(int $cinemaId, \DateTimeInterface $day): array
{
    $qb = $this->createQueryBuilder('s')
        ->join('s.room', 'r')
        ->join('r.cinema', 'c')
        ->join('s.film', 'f')
        ->addSelect('r', 'c', 'f')
        ->andWhere('c.id = :cinema')->setParameter('cinema', $cinemaId)
        ->andWhere('DATE(s.startTime) = :day')->setParameter('day', $day->format('Y-m-d'))
        ->andWhere('s.placesAvailable > 0')
        ->orderBy('s.startTime', 'ASC');

    return $qb->getQuery()->getResult();
}
```

- **Attention** : si `DATE()` n'est pas supporté par votre version SQL, utilisez `s.startTime BETWEEN :dayStart AND :dayEnd`.

3.3. `ReservationRepository`, `ReviewRepository`, `IncidentRepository`

- Implémentez, par symétrie, des méthodes pour récupérer :
  - Les réservations d'un utilisateur classées par date,
  - Les avis validés pour un film (`status = 'VALIDATED'`),
  - Les incidents ouverts d'une salle, etc.

---

4. Refonte des Contrôleurs

Tous les contrôleurs actuels utilisant React doivent être repris : ils n'exposeront plus d'API JSON pour React, mais retourneront des pages Twig. Voici l'organisation globale recommandée :

```
src/Controller/
├── DefaultController.php      # pages publiques (home, contact, liste
films, etc.)
├── SecurityController.php     # inscription, connexion, logout, reset
password
├── UserController.php         # espace "Mon Espace" (historique
réservations, noter films)
├── AdminController.php        # espace admin (gestion films, salles,
séances, employés)
├── EmployeeController.php     # espace employé (gestion salles, séances,
validation avis, incidents)
└── Api/                       # ne plus utiliser, ou uniquement pour
AJAX/Stimulus
    └── …
```

Note : nous abandonnons l'approche "API + React" ; tout sera rendu côté serveur via Twig + Stimulus.

4.1. `DefaultController` (public)

1. **Constat** :
   - Probablement un contrôleur existant qui renvoyait une vue JSON pour React.
2. **Objectif** :
   - Gérer :
     - La page d'accueil (dernier mercredi),
     - La liste des films (avec filtres),
     - La page "réservation" (choix cinéma/film/séance),
     - La page "contact" (formulaire).
3. **Actions détaillées** :

```
<?php
namespace App\Controller;

use App\Repository\FilmRepository;
use App\Repository\CinemaRepository;
use App\Repository\ScreeningRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends AbstractController
{
    private FilmRepository $filmRepo;
    private CinemaRepository $cinemaRepo;
    private ScreeningRepository $screeningRepo;

    public function __construct(
        FilmRepository $filmRepo,
        CinemaRepository $cinemaRepo,
        ScreeningRepository $screeningRepo
    ) {
        $this->filmRepo = $filmRepo;
        $this->cinemaRepo = $cinemaRepo;
```

```
28.          $this->screeningRepo = $screeningRepo;
29.      }
30.
31.      #[Route('/', name: 'home')]
32.      public function index(): Response
33.      {
34.          // Récupérer les films du dernier mercredi
35.          $lastWedFilms = $this->filmRepo->findLastWednesdayFilms();
36.          return $this->render('default/index.html.twig', [
37.              'films' => $lastWedFilms,
38.          ]);
39.      }
40.
41.      #[Route('/films', name: 'film_list')]
42.      public function filmList(Request $request): Response
43.      {
44.          $cinemaId = $request->query->getInt('cinema', 0) ?: null;
45.          $genre    = $request->query->get('genre') ?: null;
46.          $dayStr   = $request->query->get('day') ?: null;
47.          $day      = $dayStr ? new \DateTimeImmutable($dayStr) :
   null;
48.
49.          $films = $this->filmRepo->findByCinemaGenreDay($cinemaId,
   $genre, $day);
50.
51.          $cinemas = $this->cinemaRepo->findAll();
52.          $genres  = $this->filmRepo->findDistinctGenres(); //
   Méthode à implémenter dans FilmRepository
53.
54.          return $this->render('default/film_list.html.twig', [
55.              'films'   => $films,
56.              'cinemas' => $cinemas,
57.              'genres'  => $genres,
58.              'selectedCinema' => $cinemaId,
59.              'selectedGenre'  => $genre,
60.              'selectedDay'    => $day?->format('Y-m-d'),
61.          ]);
62.      }
63.
64.      #[Route('/reservation', name: 'reservation')]
65.      public function reservation(Request $request): Response
66.      {
67.          // Si l'utilisateur n'est pas connecté, rediriger vers
   login
68.          if (!$this->getUser()) {
69.              $this->addFlash('warning', 'Vous devez être connecté
   pour réserver.');
70.              return $this->redirectToRoute('app_login');
71.          }
72.
73.          // Récupérer paramètres préremplis (film, cinéma, séance)
   s'ils existent
74.          $cinemaId   = $request->query->getInt('cinema', 0) ?: null;
75.          $filmId     = $request->query->getInt('film', 0) ?: null;
76.          $screeningId= $request->query->getInt('screening', 0) ?:
   null;
77.
78.          $cinemas    = $this->cinemaRepo->findAll();
79.          $films      = $this->filmRepo->findAll(); // ou
   findAvailableInCinemas
```

```
        $screenings = $screeningId ? [$this->screeningRepo->find($screeningId)] : [];

        return $this->render('default/reservation.html.twig', [
            'cinemas'    => $cinemas,
            'films'      => $films,
            'screenings' => $screenings,
            'selectedCinema'    => $cinemaId,
            'selectedFilm'      => $filmId,
            'selectedScreening' => $screeningId,
        ]);
    }

    #[Route('/contact', name: 'contact', methods: ['GET','POST'])]
    public function contact(Request $request, \Swift_Mailer $mailer): Response
    {
        // Créer un formulaire Symfony pour contact (ContactType), ou simple handling
        $form = $this->createForm(ContactType::class);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $data = $form->getData();
            $message = (new \Swift_Message('Contact Cinéphoria'))
                ->setFrom($data['email'])
                ->setTo('contact@cinephoria.fr')
                ->setBody(
                    $this->renderView('emails/contact.html.twig', [
                        'name'    => $data['name'] ?? 'Anonyme',
                        'subject' => $data['subject'],
                        'message' => $data['message'],
                    ]),
                    'text/html'
                );
            $mailer->send($message);
            $this->addFlash('success', 'Votre message a été envoyé.');
            return $this->redirectToRoute('contact');
        }

        return $this->render('default/contact.html.twig', [
            'contactForm' => $form->createView(),
        ]);
    }
}
```

Remarques Twig/UX :

- Dans `index.html.twig`, utilisez une boucle pour afficher une grille de films, par exemple :

```
{% extends 'base.html.twig' %}

{% block title %}Accueil - Cinéphoria{% endblock %}

{% block body %}
  <h1>Films ajoutés le dernier mercredi</h1>
  <div class="grid grid-cols-3 gap-4">
    {% for film in films %}
      <div class="border p-4">
        <img src="{{ asset('build/images/posters/' ~ film.posterPath) }}" alt="{{ film.title }}">
        <h2>{{ film.title }}</h2>
```

```
        <p>Âge minimum : {{ film.ageRating }} ans</p>
        <p>Note moyenne : {{ film.averageRating }}/5</p>
        <a href="{{ path('film_show', { id: film.id }) }}"
   class="btn">Détails</a>
      </div>
    {% else %}
      <p>Aucun film trouvé.</p>
    {% endfor %}
  </div>
{% endblock %}
```

4.2. `SecurityController` (inscription, connexion, déconnexion, reset password)

1. **Constat** :
   - Le projet original utilisait JWT. Nous allons passer à l'authentification "form login" de Symfony.
2. **Objectif** :
   - Permettre :
     - Inscription (`/register`),
     - Connexion (`/login`),
     - Déconnexion (`/logout`),
     - "Mot de passe oublié" (`/reset-password`).
   - Utiliser les FormTypes Symfony (`RegistrationFormType`, `LoginFormType`, `ResetPasswordRequestFormType`, `ChangePasswordFormType`).
3. **Actions détaillées** :

4.2.1. Configuration `security.yaml`

```
security:
  # encoders (ou password_hashers) pour Symfony 6+
  password_hashers:
    App\Entity\User:
      algorithm: auto

  providers:
    app_user_provider:
      entity:
        class: App\Entity\User
        property: email

  firewalls:
    dev:
      pattern: ^/_(profiler|wdt)
      security: false

    main:
      lazy: true
      provider: app_user_provider

      form_login:
        login_path: app_login
        check_path: app_login
        csrf_token_generator: security.csrf.token_manager

      logout:
        path:   app_logout
```

```
        # où rediriger après logout
        target: app_home

      # si besoin, gérer "remember me"
      # remember_me:
      #   secret:   '%kernel.secret%'
      #   lifetime: 604800 # 1 semaine

  access_control:
    # autoriser l'accès à l'espace admin seulement aux ROLE_ADMIN
    - { path: ^/admin, roles: ROLE_ADMIN }
    # accès employé aux URI /employee
    - { path: ^/employee, roles: ROLE_EMPLOYEE }
    # accès utilisateurs connectés pour /reservation
    - { path: ^/reservation, roles: ROLE_USER }
    # page de gestion des mots de passe oubliés ouverte à tous
    - { path: ^/reset-password, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    # page d'inscription & de connexion ouvertes à tous
    - { path: ^/(login|register), roles: IS_AUTHENTICATED_ANONYMOUSLY
}
```

4.2.2. SecurityController.php

```
<?php
namespace App\Controller;

use App\Entity\User;
use App\Form\RegistrationFormType;
use App\Form\LoginFormType;
use App\Form\ResetPasswordRequestFormType;
use App\Form\ChangePasswordFormType;
use App\Repository\UserRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use
Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;
use Symfony\Component\Routing\Annotation\Route;
use
Symfony\Component\Security\Http\Authentication\AuthenticationUtils;
use SymfonyCasts\Bundle\ResetPassword\ResetPasswordHelperInterface;
use SymfonyCasts\Bundle\ResetPassword\Model\ResetPasswordRequest;
use
SymfonyCasts\Bundle\ResetPassword\Exception\ResetPasswordExceptionInt
erface;

class SecurityController extends AbstractController
{
    private EntityManagerInterface $em;
    private UserPasswordHasherInterface $passwordHasher;
    private ResetPasswordHelperInterface $resetPasswordHelper;

    public function __construct(
        EntityManagerInterface $em,
        UserPasswordHasherInterface $passwordHasher,
        ResetPasswordHelperInterface $resetPasswordHelper
    ) {
        $this->em = $em;
        $this->passwordHasher = $passwordHasher;
        $this->resetPasswordHelper = $resetPasswordHelper;
```

```
    }

    #[Route('/register', name: 'app_register', methods:
['GET','POST'])]
    public function register(Request $request): Response
    {
        if ($this->getUser()) {
            return $this->redirectToRoute('home');
        }

        $user = new User();
        $form = $this->createForm(RegistrationFormType::class,
$user);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // Hasher le mot de passe
            $user->setPassword(
                $this->passwordHasher->hashPassword(
                    $user,
                    $form->get('plainPassword')->getData()
                )
            );
            $this->em->persist($user);
            $this->em->flush();

            // éventuellement envoyer mail de confirmation, etc.
            $this->addFlash('success', 'Votre compte a été créé. Vous
pouvez vous connecter.');
            return $this->redirectToRoute('app_login');
        }

        return $this->render('security/register.html.twig', [
            'registrationForm' => $form->createView(),
        ]);
    }

    #[Route('/login', name: 'app_login', methods: ['GET','POST'])]
    public function login(AuthenticationUtils $authenticationUtils):
Response
    {
        if ($this->getUser()) {
            return $this->redirectToRoute('home');
        }

        // get the login error if there is one
        $error = $authenticationUtils->getLastAuthenticationError();
        // last username entered by the user
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render('security/login.html.twig', [
            'last_username' => $lastUsername,
            'error'         => $error,
        ]);
    }

    #[Route('/logout', name: 'app_logout')]
    public function logout(): void
    {
        // Cette méthode peut rester vide ; elle est interceptée par
la firewall logout
```

```
        throw new \Exception('This should never be reached');
    }

    #[Route('/reset-password', name: 'app_forgot_password_request',
methods: ['GET','POST'])]
    public function request(Request $request, UserRepository
$userRepository): Response
    {
        $form = $this-
>createForm(ResetPasswordRequestFormType::class);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $email = $form->get('email')->getData();
            $user = $userRepository->findOneBy(['email' => $email]);

            if (!$user) {
                $this->addFlash('danger', 'Aucun utilisateur trouvé
avec cet email.');
                return $this-
>redirectToRoute('app_forgot_password_request');
            }

            // Gérer l'envoi du mail de reset (via SymfonyCasts
ResetPasswordBundle par exemple)
            try {
                $resetToken = $this->resetPasswordHelper-
>generateResetToken($user);
            } catch (ResetPasswordExceptionInterface $e) {
                $this->addFlash('danger', 'Impossible de générer le
lien de réinitialisation.');
                return $this-
>redirectToRoute('app_forgot_password_request');
            }

            $emailMessage = (new
\Symfony\Bridge\Twig\Mime\TemplatedEmail())
                ->from('no-reply@cinephoria.fr')
                ->to($user->getEmail())
                ->subject('Votre demande de réinitialisation de mot
de passe')
                -
>htmlTemplate('security/reset_password_email.html.twig')
                ->context([
                    'resetToken' => $resetToken,
                ])
            ;
            $mailer = $this->get('mailer'); // ou injection
            $mailer->send($emailMessage);

            $this->addFlash('success', 'Un email vous a été envoyé
pour réinitialiser votre mot de passe.');
            return $this->redirectToRoute('home');
        }

        return $this->render('security/request.html.twig', [
            'requestForm' => $form->createView(),
        ]);
    }
```

```
    #[Route('/reset-password/reset/{token}', name:
'app_reset_password', methods: ['GET','POST'])]
    public function reset(Request $request, string $token = null):
Response
    {
        if ($token) {
            // stocker le token en session pour vérification lors du
submit
            $request->getSession()->set('reset_password_token',
$token);
        } else {
            $token = $request->getSession()-
>get('reset_password_token');
        }

        if (!$token) {
            throw $this->createNotFoundException('Aucun jeton de
réinitialisation fourni.');
        }

        try {
            $user = $this->resetPasswordHelper-
>validateTokenAndFetchUser($token);
        } catch (ResetPasswordExceptionInterface $e) {
            $this->addFlash('danger', 'Ce lien de réinitialisation
n'est pas valide.');
            return $this-
>redirectToRoute('app_forgot_password_request');
        }

        $form = $this->createForm(ChangePasswordFormType::class);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // effacer le token pour empêcher reuse
            $this->resetPasswordHelper->removeResetRequest($token);

            $encodedPassword = $this->passwordHasher->hashPassword(
                $user,
                $form->get('plainPassword')->getData()
            );
            $user->setPassword($encodedPassword);
            $this->em->flush();

            $this->addFlash('success', 'Votre mot de passe a été
réinitialisé.');
            return $this->redirectToRoute('app_login');
        }

        return $this->render('security/reset.html.twig', [
            'resetForm' => $form->createView(),
        ]);
    }
}
```

4. **FormTypes associés**
   - **`RegistrationFormType`** :
   - `<?php`
   - `namespace App\Form;`
   -
   - `use App\Entity\User;`

```
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\PasswordType;
use Symfony\Component\Form\Extension\Core\Type\RepeatedType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Validator\Constraints as Assert;

class RegistrationFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder,
array $options): void
    {
        $builder
            ->add('email', EmailType::class)
            ->add('firstName', TextType::class)
            ->add('lastName', TextType::class)
            ->add('plainPassword', RepeatedType::class, [
                'type' => PasswordType::class,
                'mapped' => false,
                'first_options'  => ['label' => 'Mot de
passe'],
                'second_options' => ['label' => 'Répéter le mot
de passe'],
                'constraints' => [
                    new Assert\NotBlank(),
                    new Assert\Length(['min' => 8]),
                    new Assert\Regex([
                        'pattern' => '/[A-Z]/',
                        'message' => 'Au moins une majuscule.'
                    ]),
                    new Assert\Regex([
                        'pattern' => '/[a-z]/',
                        'message' => 'Au moins une minuscule.'
                    ]),
                    new Assert\Regex([
                        'pattern' => '/\d/',
                        'message' => 'Au moins un chiffre.'
                    ]),
                    new Assert\Regex([
                        'pattern' => '/[\W]/',
                        'message' => 'Au moins un caractère
spécial.'
                    ]),
                ],
            ]);
    }

    public function configureOptions(OptionsResolver
$resolver): void
    {
        $resolver->setDefaults([
            'data_class' => User::class,
        ]);
    }
}
```

`ResetPasswordRequestFormType` (un simple champ email) :

```
<?php
namespace App\Form;

```

```
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Validator\Constraints as Assert;

class ResetPasswordRequestFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder,
array $options): void
    {
        $builder
            ->add('email', EmailType::class, [
                'constraints' => [
                    new Assert\NotBlank(),
                    new Assert\Email(),
                ],
            ]);
    }

    public function configureOptions(OptionsResolver
$resolver): void
    {
        $resolver->setDefaults([]);
    }
}
```

`ChangePasswordFormType` (idem au registration pour plainPassword).

`ContactType` (dans `DefaultController`) :

```
<?php
namespace App\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Validator\Constraints as Assert;

class ContactType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder,
array $options): void
    {
        $builder
            ->add('name', TextType::class, [
                'required' => false,
                'label' => 'Votre nom (facultatif)',
            ])
            ->add('email', EmailType::class, [
                'constraints' => [
                    new Assert\NotBlank(),
                    new Assert\Email(),
                ],
            ])
            ->add('subject', TextType::class, [
                'constraints' => [ new Assert\NotBlank() ],
            ])
            ->add('message', TextareaType::class, [
                'constraints' => [ new Assert\NotBlank() ],
            ]);
```

```
    }

    public function configureOptions(OptionsResolver
$resolver): void
    {
        $resolver->setDefaults([]);
    }
}
```

4.3. `UserController` (espace utilisateur)

1. **Constat** :
   - Il existait peut-être un contrôleur d'API pour "mes réservations" et "noter film". Nous devons le remplacer par un contrôleur standard qui affiche une page d'historique de réservation et permet de poster un avis.
2. **Objectif** :
   - Page "Mon Espace" :
     - Lister toutes les réservations (`Reservation`) de l'utilisateur triées par date ;
     - Pour chaque réservation dont la séance est passée, proposer un formulaire `ReviewType` pour noter le film (si pas déjà noté).
   - Lorsqu'un avis est soumis, créer une `Review` liée à cet utilisateur et ce film, puis re-calculez la note moyenne du film.
3. **Actions détaillées** :

```
<?php
namespace App\Controller;

use App\Entity\Review;
use App\Form\ReviewFormType;
use App\Repository\ReservationRepository;
use App\Repository\ReviewRepository;
use App\Repository\FilmRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Security\Http\Attribute\IsGranted;

#[IsGranted('ROLE_USER')]
class UserController extends AbstractController
{
    private ReservationRepository $reservationRepo;
    private FilmRepository $filmRepo;
    private ReviewRepository $reviewRepo;
    private EntityManagerInterface $em;

    public function __construct(
        ReservationRepository $reservationRepo,
        FilmRepository $filmRepo,
        ReviewRepository $reviewRepo,
        EntityManagerInterface $em
    ) {
        $this->reservationRepo = $reservationRepo;
        $this->filmRepo = $filmRepo;
        $this->reviewRepo = $reviewRepo;
        $this->em = $em;
    }
```

```
38.
39.      #[Route('/mon-espace', name: 'user_dashboard')]
40.      public function dashboard(Request $request): Response
41.      {
42.          /** @var \App\Entity\User $user */
43.          $user = $this->getUser();
44.
45.          // Récupérer toutes les réservations de l'utilisateur
46.          $reservations = $this->reservationRepo->findBy(
47.              ['user' => $user],
48.              ['createdAt' => 'DESC']
49.          );
50.
51.          // Construire pour chaque réservation un formulaire de note
    si applicable
52.          $reviewForms = [];
53.          foreach ($reservations as $reservation) {
54.              $screening = $reservation->getScreening();
55.              if ($screening->getEndTime() < new
    \DateTimeImmutable()) {
56.                  // La séance est passée, vérifier s'il y a déjà un
    avis de cet utilisateur sur ce film
57.                  $existingReview = $this->reviewRepo->findOneBy([
58.                      'user' => $user,
59.                      'film' => $screening->getFilm(),
60.                  ]);
61.
62.                  if (!$existingReview) {
63.                      $review = new Review();
64.                      $review->setUser($user)
65.                             ->setFilm($screening->getFilm());
66.                      $form = $this-
    >createForm(ReviewFormType::class, $review, [
67.                          'action' => $this-
    >generateUrl('user_add_review', ['filmId' => $screening->getFilm()-
    >getId()]),
68.                      ]);
69.                      $reviewForms[$screening->getFilm()->getId()] =
    $form->createView();
70.                  }
71.              }
72.          }
73.
74.          return $this->render('user/dashboard.html.twig', [
75.              'reservations' => $reservations,
76.              'reviewForms'  => $reviewForms,
77.          ]);
78.      }
79.
80.      #[Route('/mon-espace/ajouter-avis/{filmId}', name:
    'user_add_review', methods: ['POST'])]
81.      public function addReview(int $filmId, Request $request):
    Response
82.      {
83.          $user = $this->getUser();
84.          $film = $this->filmRepo->find($filmId);
85.          if (!$film) {
86.              throw $this->createNotFoundException('Film
    introuvable.');
87.          }
88.
```

```
89.          // Vérifier qu'il n'y a pas déjà un avis
90.          $existing = $this->reviewRepo->findOneBy(['user' => $user,
   'film' => $film]);
91.          if ($existing) {
92.              $this->addFlash('warning', 'Vous avez déjà noté ce
   film.');
93.              return $this->redirectToRoute('user_dashboard');
94.          }
95.
96.          $review = new Review();
97.          $review->setUser($user)
98.                 ->setFilm($film);
99.          $form = $this->createForm(ReviewFormType::class, $review);
100.         $form->handleRequest($request);
101.
102.         if ($form->isSubmitted() && $form->isValid()) {
103.             $this->em->persist($review);
104.             // Recalculer la note moyenne du film
105.             $film->recalcAverageRating();
106.             $this->em->flush();
107.
108.             $this->addFlash('success', 'Merci pour votre avis, il
   sera validé prochainement.');
109.             return $this->redirectToRoute('user_dashboard');
110.         }
111.
112.         $this->addFlash('danger', 'Formulaire invalide.');
113.         return $this->redirectToRoute('user_dashboard');
114.     }
115. }
```

116. **ReviewFormType** :

```
117. <?php
118. namespace App\Form;
119.
120. use App\Entity\Review;
121. use Symfony\Component\Form\AbstractType;
122. use Symfony\Component\Form\FormBuilderInterface;
123. use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
124. use Symfony\Component\Form\Extension\Core\Type\TextareaType;
125. use Symfony\Component\OptionsResolver\OptionsResolver;
126. use Symfony\Component\Validator\Constraints as Assert;
127.
128. class ReviewFormType extends AbstractType
129. {
130.     public function buildForm(FormBuilderInterface $builder, array
   $options): void
131.     {
132.         $builder
133.             ->add('rating', ChoiceType::class, [
134.                 'choices'  => [
135.                     '1 étoile' => 1,
136.                     '2 étoiles' => 2,
137.                     '3 étoiles' => 3,
138.                     '4 étoiles' => 4,
139.                     '5 étoiles' => 5,
140.                 ],
141.                 'constraints' => [ new Assert\NotBlank() ],
142.             ])
143.             ->add('comment', TextareaType::class, [
144.                 'required' => false,
145.                 'attr' => ['rows' => 3],
```

```
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver):
  void
    {
        $resolver->setDefaults([
            'data_class' => Review::class,
        ]);
    }
}
```

Vues Twig :

- Dans `templates/user/dashboard.html.twig` affichez la liste des réservations et insérez, à la bonne place, les formulaires de notation (Stimulus non nécessaire ici, un simple Twig suffit).
- Extrait possible :

```
{% extends 'base.html.twig' %}
{% block title %}Mon Espace{% endblock %}
{% block body %}
  <h1>Mon Espace</h1>
  <h2>Mes réservations</h2>
  <table class="w-full border-collapse">
    <thead>
      <tr>
        <th>Film</th>
        <th>Salle</th>
        <th>Heure</th>
        <th>Places</th>
        <th>Statut</th>
        <th>Note</th>
      </tr>
    </thead>
    <tbody>
      {% for res in reservations %}
        <tr>
          <td>{{ res.screening.film.title }}</td>
          <td>{{ res.screening.room.number }}</td>
          <td>{{ res.screening.startTime|date('d/m/Y H:i')
}}</td>
          <td>{{ res.seatsBooked }}</td>
          <td>{{ res.status }}</td>
          <td>
            {% if reviewForms[res.screening.film.id] is defined
%}
              {{ form_start(reviewForms[res.screening.film.id])
}}
                {{
form_widget(reviewForms[res.screening.film.id].rating) }}
                {{
form_widget(reviewForms[res.screening.film.id].comment) }}
                <button type="submit" class="btn btn-primary
btn-sm">Noter</button>
              {{ form_end(reviewForms[res.screening.film.id])
}}
            {% else %}
              {% set existingReview = app.user.reviews|filter(r
=> r.film.id == res.screening.film.id)|first %}
              {% if existingReview %}
                {{ existingReview.rating }}/5
```

```
                {% else %}
                  (en attente)
                {% endif %}
              {% endif %}
            </td>
          </tr>
        {% endfor %}
      </tbody>
    </table>
{% endblock %}
```

4.4. `AdminController` (Espace administrateur)

1. **Constat** :
   - Actuellement, les admin ont potentiellement un ensemble d'API, stockées en `src/Controller/Api/Admin…`. Nous allons tout remplacer par un seul contrôleur `AdminController.php`affichant des pages Twig, et proposant des formulaires Symfony (via `make:crud` si souhaité).
2. **Objectif** :
   - Gérer :
     1. **Films** : Créer / Modifier / Supprimer,
     2. **Salles** : Créer / Modifier / Supprimer,
     3. **Séances** : Créer / Modifier / Supprimer,
     4. **Employés** (création de compte employé, reset mot de passe),
     5. **Dashboard** (nombre de réservations par film sur 7 jours, via MongoDB).
3. **Actions détaillées** :

4.4.1. Gestion des films, salles, séances

- Utilisez la commande `make:crud` pour générer rapidement les entités :

```
php bin/console make:crud Film
php bin/console make:crud Room
php bin/console make:crud Screening
```

- Cela génère des pages d'administration Twig dans `templates/film/`, `templates/room/`, `templates/screening/`, ainsi que le contrôleur `FilmController.php`, `RoomController.php`, `ScreeningController.php`.
  - **Note** : Concernant la branche `refacto/symfony-ux`, vous pouvez renommer `FilmController` en `AdminController` et regrouper les routes sous `/admin/film`, `/admin/room`, `/admin/screening`.
  - Ajustez les **annotations** ou **attributs** de route pour préfixer "`/admin/…`" et ajoutez `#[IsGranted('ROLE_ADMIN')]` au-dessus de chaque action.
- Par exemple, dans `src/Controller/FilmController.php` (à renommer en `AdminController.php`), modifiez :

```
#[Route('/admin/film', name: 'film_index', methods: ['GET'])]
#[IsGranted('ROLE_ADMIN')]
public function index(FilmRepository $filmRepository): Response
{
    return $this->render('admin/film/index.html.twig', [
        'films' => $filmRepository->findAll(),
```

```
    ]);
}
```

- Répétez pour `new`, `edit`, `delete` (routes POST/GET correspondantes).
- Pour les salles et séances, faites de même.

4.4.2. Création de comptes employés & reset de mot de passe

- Ajoutez dans `AdminController.php` une action pour créer un utilisateur avec rôle `ROLE_EMPLOYEE` :

```
#[Route('/admin/employee/new', name: 'admin_employee_new', methods: ['GET','POST'])]
public function newEmployee(Request $request): Response
{
    $user = new User();
    $user->setRoles(['ROLE_EMPLOYEE']);
    $form = $this->createForm(RegistrationFormType::class, $user);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $user->setPassword(
            $this->passwordHasher->hashPassword(
                $user,
                $form->get('plainPassword')->getData()
            )
        );
        $this->em->persist($user);
        $this->em->flush();

        $this->addFlash('success', 'Employé créé.');
        return $this->redirectToRoute('admin_employee_index');
    }

    return $this->render('admin/employee/new.html.twig', [
        'employeeForm' => $form->createView(),
    ]);
}
```

- Pour réinitialiser le mot de passe d'un employé, vous pouvez rediriger vers la logique de "mot de passe oublié" en lui générant un token, ou forcer un nouveau mot de passe temporaire et l'afficher (ou l'envoyer par mail).

4.4.3. Dashboard – statistiques MongoDB

- Étant donné que vous utilisez MongoDB pour stocker les statistiques, dans `AdminController.php`, créez une action `/admin/dashboard` :

```
#[Route('/admin/dashboard', name: 'admin_dashboard', methods: ['GET'])]
public function dashboard(\MongoDB\Client $mongoClient): Response
{
    $db = $mongoClient->selectDatabase('cinephoria_stats');
    $collection = $db->selectCollection('reservations_stats');

    // Supposons que chaque document contient { filmId, date: ISODate, count: int }
    // On veut les 7 derniers jours
    $today = new \DateTimeImmutable();
    $weekAgo = $today->modify('-7 days')->setTime(0,0,0);
```

```
        $pipeline = [
            [
                '$match' => [
                    'date' => [
                        '$gte' => new
\MongoDB\BSON\UTCDateTime($weekAgo->getTimestamp()*1000),
                        '$lte' => new
\MongoDB\BSON\UTCDateTime($today->getTimestamp()*1000)
                    ]
                ]
            ],
            [
                '$group' => [
                    '_id' => ['filmId' => '$filmId', 'day' =>
['$dateToString' => ['format'=>'%Y-%m-%d','date'=>'$date']]],
                    'total' => ['$sum' => '$count']
                ]
            ],
            [
                '$sort' => ['_id.day' => 1]
            ]
        ];
        $statsCursor = $collection->aggregate($pipeline)-
>toArray();

        // Organiser les données pour Twig (tableau filmId → [
jour1=>count1, ... ])
        $stats = [];
        foreach ($statsCursor as $doc) {
            $filmId = $doc->_id->filmId;
            $day    = $doc->_id->day;
            $stats[$filmId][$day] = $doc->total;
        }

        // Récupérer la liste des films correspondants
        $filmRepo = $this->filmRepo;
        $films = $filmRepo->findBy(['id' => array_keys($stats)]);

        return $this->render('admin/dashboard.html.twig', [
            'stats' => $stats,
            'films' => $films,
            'dates' => array_map(fn($d) => (new
\DateTimeImmutable($d))->format('d/m Y'),
array_keys(reset($stats))),
        ]);
    }
```

- Dans Twig (`templates/admin/dashboard.html.twig`), affichez un tableau avec les jours en colonnes et les films en lignes.

4. **Sécurisation**
    - Assurez-vous que chaque route administrateur est protégée par `#[IsGranted('ROLE_ADMIN')]`.

4.5. `EmployeeController` (Espace employé)

1. **Constat** :
    - Les employés doivent pouvoir gérer :

- Création/édition/suppression de films, salles, séances (idem admin, ou éventuellement restreint),
- Valider ou supprimer les avis (passe le status de “PENDING” à “VALIDATED” ou “REJECTED”),
- Saisir des incidents dans une séance (relation vers `Room`),
- Accéder à un dashboard simplifié.

2. **Objectif** :
   - Regrouper les actions sous `/employee/...`, avec `#[IsGranted('ROLE_EMPLOYEE')]`.
   - S’assurer que la validation d’avis repose sur une route POST qui change le `status` d’un `Review`.
   - Permettre la création d’`Incident` via un formulaire `IncidentFormType`.
3. **Actions détaillées** :

4.5.1. Validation / suppression d’avis

```
<?php
namespace App\Controller;

use App\Repository\ReviewRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\RedirectResponse;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Security\Http\Attribute\IsGranted;

#[IsGranted('ROLE_EMPLOYEE')]
class EmployeeController extends AbstractController
{
    private ReviewRepository $reviewRepo;
    private EntityManagerInterface $em;

    public function __construct(ReviewRepository $reviewRepo,
EntityManagerInterface $em)
    {
        $this->reviewRepo = $reviewRepo;
        $this->em = $em;
    }

    #[Route('/employee/reviews', name: 'employee_review_list',
methods: ['GET'])]
    public function reviewList(): Response
    {
        $pendingReviews = $this->reviewRepo->findBy(['status' =>
'PENDING']);
        return $this->render('employee/review_list.html.twig', [
            'pendingReviews' => $pendingReviews,
        ]);
    }

    #[Route('/employee/review/{id}/validate', name:
'employee_review_validate', methods: ['POST'])]
    public function validateReview(int $id): RedirectResponse
    {
        $review = $this->reviewRepo->find($id);
        if (!$review) {
```

```
            $this->addFlash('danger', 'Avis introuvable.');
        } else {
            $review->setStatus('VALIDATED');
            $this->em->flush();

            // Recalculer la note moyenne du film
            $review->getFilm()->recalcAverageRating();
            $this->em->flush();

            $this->addFlash('success', 'Avis validé.');
        }
        return $this->redirectToRoute('employee_review_list');
    }

    #[Route('/employee/review/{id}/reject', name:
'employee_review_reject', methods: ['POST'])]
    public function rejectReview(int $id): RedirectResponse
    {
        $review = $this->reviewRepo->find($id);
        if (!$review) {
            $this->addFlash('danger', 'Avis introuvable.');
        } else {
            $review->setStatus('REJECTED');
            $this->em->flush();
            $this->addFlash('success', 'Avis rejeté.');
        }
        return $this->redirectToRoute('employee_review_list');
    }
}
```

- **Vue Twig** (`templates/employee/review_list.html.twig`) : listez chaque avis en attente avec un formulaire POST pour “Valider” et “Rejeter”.

4.5.2. Gestion des incidents

```
<?php
namespace App\Controller;

use App\Entity\Incident;
use App\Form\IncidentFormType;
use App\Repository\RoomRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Security\Http\Attribute\IsGranted;

#[IsGranted('ROLE_EMPLOYEE')]
class EmployeeController extends AbstractController
{
    private RoomRepository $roomRepo;
    private EntityManagerInterface $em;

    public function __construct(RoomRepository $roomRepo,
EntityManagerInterface $em)
    {
        $this->roomRepo = $roomRepo;
        $this->em = $em;
    }
```

```
    #[Route('/employee/incident/new', name: 'employee_incident_new',
methods: ['GET','POST'])]
    public function newIncident(Request $request): Response
    {
        $incident = new Incident();
        $incident->setReporter($this->getUser());
        $form = $this->createForm(IncidentFormType::class,
$incident);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $this->em->persist($incident);
            $this->em->flush();
            $this->addFlash('success', 'Incident signalé.');
            return $this->redirectToRoute('employee_incident_list');
        }

        return $this->render('employee/incident/new.html.twig', [
            'incidentForm' => $form->createView(),
        ]);
    }

    #[Route('/employee/incidents', name: 'employee_incident_list',
methods: ['GET'])]
    public function listIncidents(): Response
    {
        $incidents = $this->em->getRepository(Incident::class)-
>findBy([], ['createdAt' => 'DESC']);
        return $this->render('employee/incident/list.html.twig', [
            'incidents' => $incidents,
        ]);
    }
}
```

- **IncidentFormType** :
- <?php
- namespace App\Form;
-
- use App\Entity\Incident;
- use App\Entity\Room;
- use Symfony\Bridge\Doctrine\Form\Type\EntityType;
- use Symfony\Component\Form\AbstractType;
- use Symfony\Component\Form\FormBuilderInterface;
- use Symfony\Component\Form\Extension\Core\Type\TextareaType;
- use Symfony\Component\OptionsResolver\OptionsResolver;
- use Symfony\Component\Validator\Constraints as Assert;
-
- class IncidentFormType extends AbstractType
- {
- public function buildForm(FormBuilderInterface $builder,
  array $options): void
- {
- $builder
- ->add('room', EntityType::class, [
- 'class' => Room::class,
- 'choice_label' => fn(Room $r) => $r-
  >getCinema()->getName() . ' - Salle ' . $r->getNumber(),
-])
- ->add('description', TextareaType::class, [
- 'constraints' => [new Assert\NotBlank()],

```
            ])
        ;
    }

    public function configureOptions(OptionsResolver
$resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Incident::class,
        ]);
    }
}
```

- **Vue Twig** (`templates/employee/incident/new.html.twig`) : formulaire simple, affichage de la liste.

---

5. Refonte des Vues Twig et intégration Stimulus (Symfony UX)

5.1. Organisation globale des templates

- `templates/base.html.twig` :
  - Contiendra le squelette HTML commun (doctype, head, liens CSS via Asset Mapper, inclusion Stimulus, `<header>` avec menu, `<footer>`).
  - Exemple :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>{% block title %}Cinéphoria{% endblock %}</title>
    <link rel="stylesheet" href="{{ asset('build/css/app.css')
}}">
    <script src="{{ asset('build/js/controllers.js') }}"
defer></script>
  </head>
  <body>
    <header class="bg-gray-800 text-white p-4">
      <div class="container mx-auto flex justify-between">
        <a href="{{ path('home') }}" class="font-bold text-
xl">Cinéphoria</a>
        <nav>
          <ul class="flex space-x-4">
            <li><a href="{{ path('home') }}">Accueil</a></li>
            <li><a href="{{ path('film_list')
}}">Films</a></li>
            <li><a href="{{ path('reservation')
}}">Réservation</a></li>
            <li><a href="{{ path('contact')
}}">Contact</a></li>
            {% if app.user %}
              <li><a href="{{ path('user_dashboard') }}">Mon
Espace</a></li>
              {% if is_granted('ROLE_EMPLOYEE') %}
                <li><a href="{{ path('employee_review_list')
}}">Espace Employé</a></li>
              {% endif %}
```

```
                {% if is_granted('ROLE_ADMIN') %}
                  <li><a href="{{ path('admin_dashboard')
}}">Admin</a></li>
                {% endif %}
                <li><a href="{{ path('app_logout')
}}">Déconnexion</a></li>
              {% else %}
                <li><a href="{{ path('app_login')
}}">Connexion</a></li>
                <li><a href="{{ path('app_register')
}}">Inscription</a></li>
              {% endif %}
            </ul>
          </nav>
        </div>
      </header>

      <main class="container mx-auto py-6">
        {% for label, messages in app.flashes %}
          <div class="alert alert-{{ label }}">
            {% for message in messages %} {{ message }}{% endfor
%}
          </div>
        {% endfor %}
        {% block body %}{% endblock %}
      </main>

      <footer class="bg-gray-200 text-gray-700 p-4 text-center">
        <p>Adresse du cinéma, téléphone, horaires…</p>
      </footer>
    </body>
  </html>
```

5.2. Utilisation de Stimulus pour comportements UI légers

- Créez dans `assets/js/controllers` des contrôleurs Stimulus lorsque vous avez besoin d'un comportement côté client (ex : choix dynamique d'un film-> charger séances, calcul instantané du prix total, etc.).
- Exemple : un contrôleur `ReservationController` pour afficher dynamiquement les séances en fonction du cinéma et du film sélectionnés :

```
// assets/js/controllers/reservation_controller.js
import { Controller } from '@hotwired/stimulus';

// Connects to data-controller="reservation"
export default class extends Controller {
  static targets = ["cinema", "film", "screening", "price", "seats"];

  connect() {
    // init
  }

  fetchScreenings() {
    const cinemaId = this.cinemaTarget.value;
    const filmId   = this.filmTarget.value;
    // appeler un endpoint Symfony qui renvoie en JSON les séances
(REST minimal)
    fetch(`/api/screenings?cinema=${cinemaId}&film=${filmId}`)
```

```
        .then(response => response.json())
        .then(data => {
          // remplir le select des séances
          let options = '<option value="">Choisir une séance</option>';
          data.forEach(s => {
            options += `<option value="${s.id}" data-
price='${s.price}'>${s.startTime} - ${s.price}€</option>`;
          });
          this.screeningTarget.innerHTML = options;
        });
    }

    updatePrice() {
      const option = this.screeningTarget.selectedOptions[0];
      const basePrice = option ? parseFloat(option.dataset.price) : 0;
      const seatsCount = parseInt(this.seatsTarget.value) || 0;
      const total = (basePrice * seatsCount).toFixed(2);
      this.priceTarget.textContent = `${total} €`;
    }
}
```

Dans Twig (ex. `templates/default/reservation.html.twig`), liez ce contrôleur :

```
{% block body %}
  <h1>Réservation</h1>
  <div data-controller="reservation">
    <label>Cinéma</label>
    <select data-reservation-target="cinema" data-action="change-
>reservation#fetchScreenings">
      <option value="">-- Choisir --</option>
      {% for cinema in cinemas %}
        <option value="{{ cinema.id }}" {% if selectedCinema ==
cinema.id %}selected{% endif %}>{{ cinema.name }}</option>
      {% endfor %}
    </select>

    <label>Film</label>
    <select data-reservation-target="film" data-action="change-
>reservation#fetchScreenings">
      <option value="">-- Choisir --</option>
      {% for film in films %}
        <option value="{{ film.id }}" {% if selectedFilm == film.id
%}selected{% endif %}>{{ film.title }}</option>
      {% endfor %}
    </select>

    <label>Séance</label>
    <select data-reservation-target="screening" data-action="change-
>reservation#updatePrice">
      {% for s in screenings %}
        <option value="{{ s.id }}" data-price="{{ s.prices['4K']
}}">{{ s.startTime|date('d/m/Y H:i') }} - {{ s.prices['4K']
}}€</option>
      {% endfor %}
    </select>

```

```
    <label>Nombre de places</label>
    <input type="number" min="1" max="10" data-reservation-target="seats" data-action="input->reservation#updatePrice">

    <p>Total : <span data-reservation-target="price">0 €</span></p>

    <button class="btn btn-primary">Valider la réservation</button>
  </div>
{% endblock %}
```

- **Attention** : il faut créer un endpoint `/api/screenings` minimisé dans un contrôleur séparé, par exemple `Api/ScreeningApiController.php`, qui retourne un JSON des séances filtrées. Vous pouvez sécuriser cet endpoint en le limitant à l'accès authentifié si besoin, mais comme il ne publie que des données publiques (heures et prix), c'est facultatif.

5.3. Correction des templates existants

- **Supprimez** toute inclusion `<div id="root"></div>` ou `app.js`/`react` dans vos templates.
- **Convertissez** les pages React en pages Twig :
  - Liste des films (cf. `film_list.html.twig` plus haut).
  - Détails d'un film (`/films/{id}`) :

```
#[Route('/films/{id}', name: 'film_show', methods: ['GET'])]
public function show(int $id): Response
{
    $film = $this->filmRepo->find($id);
    if (!$film) {
        throw $this->createNotFoundException('Film introuvable.');
    }
    // Récupérer uniquement les séances futures
    $now = new \DateTimeImmutable();
    $screenings = array_filter(
        $film->getScreenings()->toArray(),
        fn($s) => $s->getStartTime() > $now
    );
    return $this->render('default/film_show.html.twig', [
        'film' => $film,
        'screenings' => $screenings,
    ]);
}
```

Twig (`templates/default/film_show.html.twig`) :

```
{% extends 'base.html.twig' %}
{% block title %}{{ film.title }}{% endblock %}
{% block body %}
  <h1>{{ film.title }}</h1>
  <img src="{{ asset('build/images/posters/' ~ film.posterPath) }}" alt="{{ film.title }}">
  <p>{{ film.description }}</p>
  <p>Âge minimum : {{ film.ageRating }} ans</p>
  <p>Note moyenne : {{ film.averageRating }}/5</p>
  {% if film.isFeatured %}
    <p><strong>Coup de cœur de l'équipe !</strong></p>
  {% endif %}
```

```
    <h2>Prochaines séances</h2>
    <ul>
      {% for s in screenings %}
        <li>
          {{ s.startTime|date('d/m/Y H:i') }} – Salle {{
s.room.number }} – Prix par place :
          {% for key, val in s.prices %}{{ key }}: {{ val }}€ {%
endfor %}
          <a href="{{ path('reservation', {
            cinema: s.room.cinema.id,
            film: film.id,
            screening: s.id
          }) }}">Réserver</a>
        </li>
      {% else %}
        <li>Aucune séance à venir.</li>
      {% endfor %}
    </ul>
{% endblock %}
```

---

6. Mise en place de PHPStan (analyse statique) et PHPUnit (tests unitaires)

6.1. Installation PHPStan

1. **Ajouter PHPStan via Composer** :
2. `composer require --dev phpstan/phpstan phpstan/extension-installer phpstan/phpstan-doctrine`
3. **Créer un fichier de configuration** `phpstan.neon.dist` à la racine :

```
parameters:
  level: max                  # niveau d'analyse, de 0 (faible) à max (très strict)
  paths:
    - src
    - tests
  autoload_files:
     - %rootDir%/../../../vendor/autoload.php
   autoload_directories:
     - %rootDir%/../../../src
   checkMissingIterableValueType: true
   checkGenericClassInNonGenericObjectType: true
   checkMissingCallableSignature: true
includes:
   - vendor/phpstan/phpstan-doctrine/extension.neon
```

18. **Exécuter PHPStan** :
19. `vendor/bin/phpstan analyse`
20. **Corriger les erreurs** :
    - Ajustez les annotations de type dans vos entités et services si besoin.
    - Si nécessaire, ajoutez des `@phpstan-ignore-next-line` pour des cas très spécifiques.

6.2. Installation PHPUnit

1. **Ajouter PHPUnit** :
2. `composer require --dev phpunit/phpunit symfony/phpunit-bridge`

3. **Création du répertoire `tests/`** à la racine du projet si inexistant.
4. **Exemple de test unitaire pour une entité** (tests/Entity/FilmTest.php) :

```php
<?php
namespace App\Tests\Entity;

use App\Entity\Film;
use PHPUnit\Framework\TestCase;

class FilmTest extends TestCase
{
    public function testRecalcAverageRatingNoReviews()
    {
        $film = new Film();
        $film->setAverageRating(0);
        // simuler aucune review
        $this->assertEquals(0.0, $film->getAverageRating());

        $film->recalcAverageRating();
        $this->assertEquals(0.0, $film->getAverageRating());
    }

    public function testRecalcAverageRatingWithReviews()
    {
        $film = new Film();
        // créer 2 reviews fictifs
        $reflection = new \ReflectionClass($film);
        $reviewsProp = $reflection->getProperty('reviews');
        $reviewsProp->setAccessible(true);
        $reviewsProp->setValue($film, new \Doctrine\Common\Collections\ArrayCollection());

        // crée des objets Review simplifiés
        $review1 = $this->createMock(\App\Entity\Review::class);
        $review1->method('getRating')->willReturn(4);
        $review2 = $this->createMock(\App\Entity\Review::class);
        $review2->method('getRating')->willReturn(2);

        $film->getReviews()->add($review1);
        $film->getReviews()->add($review2);

        $film->recalcAverageRating();
        $this->assertEquals(3.0, $film->getAverageRating());
    }
}
```

6. **Exemple de test fonctionnel pour un contrôleur** (tests/Controller/DefaultControllerTest.php) :

```php
<?php
namespace App\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class DefaultControllerTest extends WebTestCase
{
    public function testHomePageAccessible()
    {
        $client = static::createClient();
        $client->request('GET', '/');
        $this->assertResponseIsSuccessful();
        $this->assertSelectorTextContains('h1', 'Films ajoutés le dernier mercredi');
```

```
    }

    public function testFilmListPage()
    {
        $client = static::createClient();
        $client->request('GET', '/films');
        $this->assertResponseIsSuccessful();
        $this->assertSelectorExists('select[name="cinema"]');
        $this->assertSelectorExists('select[name="genre"]');
    }
}
```

lancer PHPUnit :

```
php bin/phpunit
```

---

7. Configuration de la Qualité du code (PSR-12, PHPStan, PHP-CS-Fixer)

7.1. PHP-CS-Fixer (facultatif mais recommandé)

1. **Installer** :

```
composer require --dev friendsofphp/php-cs-fixer
```

2. **Créer un fichier `.php-cs-fixer.dist.php`** à la racine :

```
<?php

$finder = PhpCsFixer\Finder::create()
    ->in(__DIR__ . '/src')
    ->in(__DIR__ . '/tests')
    ->exclude('var')
     ->exclude('vendor');

return (new PhpCsFixer\Config())
    ->setRules([
        '@PSR12'            => true,
        'array_syntax'      => ['syntax' => 'short'],
        'no_unused_imports' => true,
        'ordered_imports'   => ['sort_algorithm' => 'alpha'],
        'single_quote'      => true,
        'binary_operator_spaces' => [
            'default' => 'align_single_space_minimal',
        ],
    ])
    ->setFinder($finder);
```

3. **Exécuter** :

```
vendor/bin/php-cs-fixer fix --diff --verbose
```

---

8. CI/CD avec GitHub Actions, Docker et déploiement sur Heroku

8.1. Dockerisation

1. **Créer un `Dockerfile`** à la racine du projet :

```
# Étape 1 : builder
```

```
FROM php:8.3-fpm AS builder

# Installer les dépendances système
RUN apt-get update && apt-get install -y \
    git unzip libicu-dev libpq-dev \
    libzip-dev zip unzip \
    nodejs npm zlib1g-dev \
     && docker-php-ext-install pdo pdo_pgsql intl zip

# Installer Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

WORKDIR /app

# Copier les fichiers composer.json et composer.lock
COPY composer.json composer.lock ./
RUN composer install --no-dev --optimize-autoloader

# Copier le reste du projet
COPY . ./

# Build assets via esbuild (optionnel)
RUN npm install --prefix assets
RUN npx esbuild assets/js/controllers/index.js --bundle --
outfile=public/build/js/controllers.js --minify
RUN npm run build --prefix assets

# Étape 2 : production
FROM php:8.3-fpm

WORKDIR /app

COPY --from=builder /app /app

# Appliquer les permissions
RUN chown -R www-data:www-data /app/var

EXPOSE 9000

CMD ["php-fpm"]
```

Créer un `docker-compose.yaml` pour le dev avec Postgres + Mongo :

```
version: '3.8'

services:
  php:
    build:
      context: .
      target: builder
    volumes:
      - .:/app
    working_dir: /app
    ports:
      - "9000:9000"

  nginx:
    image: nginx:alpine
    ports:
      - "8080:80"
    depends_on:
      - php
    volumes:
```

```
      - .:/app
      - ./docker/nginx/default.conf:/etc/nginx/conf.d/default.conf

  db:
    image: postgres:14
    environment:
      POSTGRES_USER: db_user
      POSTGRES_PASSWORD: db_pass
      POSTGRES_DB: cinephoria_dev
    ports:
      - "5432:5432"

  mongo:
    image: mongo:6
    ports:
      - "27017:27017"
```

79. **Configurer Nginx (**`docker/nginx/default.conf`**)** :

```
server {
    listen 80;
    server_name localhost;
    root /app/public;

    location / {
        try_files $uri /index.php$is_args$args;
    }

    location ~ \.php$ {
        fastcgi_pass php:9000;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }

    location ~ /\.ht {
        deny all;
    }
}
```

101. **Démarrer l'environnement** :

```
docker-compose up -d
```

103. **Accéder à l'application** via `http://localhost:8080`.

8.2. GitHub Actions

1. **Créer** **`.github/workflows/ci.yaml`** :

```
name: CI

on:
  push:
    branches: [ DEV, main ]
  pull_request:
    branches: [ DEV, main ]

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    services:
```

```
15.        postgres:
16.          image: postgres:14
17.          ports:
18.            - 5432:5432
19.          env:
20.            POSTGRES_USER: db_user
21.            POSTGRES_PASSWORD: db_pass
22.            POSTGRES_DB: cinephoria_test
23.          options: >-
24.            --health-cmd="pg_isready -U db_user"
25.            --health-interval=10s
26.            --health-timeout=5s
27.            --health-retries=5
28.
29.        mongo:
30.          image: mongo:6
31.          ports:
32.            - 27017:27017
33.          options: >-
34.            --health-cmd="mongo --eval 'db.adminCommand(\"ping\")'"
35.            --health-interval=10s
36.            --health-timeout=5s
37.            --health-retries=5
38.
39.      steps:
40.        - uses: actions/checkout@v3
41.
42.        - name: Set up PHP
43.          uses: shivammathur/setup-php@v2
44.          with:
45.            php-version: 8.3
46.            extensions: pdo_pgsql, intl, bcmath, zip, mbstring, xml
47.            coverage: xdebug
48.
49.        - name: Install dependencies
50.          run: |
51.            composer install --no-interaction --prefer-dist --
   optimize-autoloader
52.
53.        - name: Create test .env
54.          run: |
55.            cp .env .env.test
56.            echo
   "DATABASE_URL=postgresql://db_user:db_pass@127.0.0.1:5432/cinephoria_
   test" >> .env.test
57.            echo "MONGO_DSN=mongodb://localhost:27017" >> .env.test
58.
59.        - name: Run migrations
60.          run: |
61.            php bin/console doctrine:database:create --env=test
62.            php bin/console doctrine:migrations:migrate --no-
   interaction --env=test
63.
64.        - name: Run PHPUnit
65.          run: |
66.            php bin/phpunit --testdox
67.
68.        - name: Run PHPStan
69.          run: |
70.            vendor/bin/phpstan analyse -c phpstan.neon
71.
```

```
      - name: PHP-CS-Fixer
        run: |
          vendor/bin/php-cs-fixer fix --dry-run --diff
```

75. **Explications** :
    - `services.postgres` et `services.mongo` démarrent des conteneurs Postgres et Mongo pour la base de test,
    - `setup-php` installe PHP 8.3 et extensions requises,
    - On copie `.env` en `.env.test` pour configurer la connexion à la base de test,
    - On exécute les migrations en environnement test,
    - On lance `phpunit`, `phpstan`, `php-cs-fixer` (dry-run pour vérifier la conformité).

8.3. Déploiement sur Heroku

1. **Procfile** à la racine du projet :

```
web: vendor/bin/heroku-php-apache2 public/
```

3. **Configurer Heroku** (en local) :

```
heroku login
heroku create cinephoria-app
heroku addons:create heroku-postgresql:hobby-dev
heroku addons:create mongolab:sandbox
```

8. **Variables d'environnement** sur Heroku (via `heroku config:set`) :

```
heroku config:set DATABASE_URL="postgres://user:pass@ec2-xx.compute-1.amazonaws.com:5432/dbname"
heroku config:set MONGO_DSN="mongodb://username:password@ds012345.mlab.com:56789/cinephoria_prod"
heroku config:set APP_ENV=prod
heroku config:set APP_SECRET="votre_phrase_secrete"
heroku config:set MAILER_DSN="smtp://user:pass@smtp.sendgrid.net:587"
```

14. **Push sur Heroku** :

```
git push heroku refacto/symfony-ux:main
```

16. **Exécuter les migrations en remote** :

```
heroku run php bin/console doctrine:migrations:migrate --no-interaction
```

---

9. Documentation à fournir pour le jury

9.1. README.md

- Décrire clairement :
    1. **Pré-requis** (Docker, PHP 8.3, Composer, Heroku CLI),
    2. **Installation en local** (commande `composer install`, config `.env`, `docker-compose up`, etc.),
    3. **Exécution des migrations**,
    4. **Accès aux pages principales**,
    5. **Utilisation des rôles** (ex : login/admin : `admin@cinephoria.fr`, mdp : `Admin123!`, etc.),
    6. **Lancer les tests** (`php bin/phpunit`, `vendor/bin/phpstan analyse`, `vendor/bin/php-cs-fixer fix --dry-run`),
    7. **Déploiement Heroku** (Procfile, variables env).

9.2. Manuel d'utilisation (PDF)

- Présenter les différents rôles :
  - **Visiteur** : page d'accueil, liste films, réservation (avec compte), contact, mot de passe oublié, inscription, etc.
  - **Utilisateur** : consulter "Mon Espace" (historique, notation).
  - **Employé** : gérer incidents, valider avis, accès intranet (gestion salles, séances).
  - **Administrateur** : gérer films, salles, séances, employés, voir le dashboard, etc.
- Donnez un exemple de parcours pas à pas pour chaque rôle.

9.3. Charte graphique (PDF)

- **Palette de couleurs** :
  - Couleur primaire (`#1F2937` – gris foncé),
  - Couleur secondaire (`#3B82F6` – bleu),
  - Couleur accent (`#FBBF24` – jaune).
- **Typographie** :
  - Police principale : "Inter", sans-serif.
- **Maquettes (wireframes + mockups)**:
  - Page d'accueil, liste films, page film, réservation, espace utilisateur, espace employé, espace admin, etc.
  - Exportez depuis Figma ou un outil de votre choix.

9.4. Document de gestion de projet (PDF)

- Expliquez la méthodologie **Scrum/Kanban** utilisée :
  - Présentez le **Backlog**, le **Sprint Backlog**, le Kanban final,
  - Donnez des captures d'écran du board Jira (ou Trello) avec colonnes : "À faire", "En cours", "Terminé", "Merge Dawn DEV/Main".
- Détaillez les tâches phasées par priorité (US, BU) selon le référentiel CDA.

9.5. Documentation technique (PDF)

1. **Architecture logicielle**
   - Structure générale en couches MVC,
   - Description des bundles Symfony UX, Asset Mapper, Doctrine, …
2. **Réflexion initiale**
   - Justification du choix Symfony 7 (PHP 8.3), PostgreSQL, MongoDB, Stimulus (UX), etc.
3. **Configuration de l'environnement**
   - Docker / docker-compose,
   - Variables `.env`,
   - Installation Composer, Node (pour Stimulus), etc.
4. **Modèle conceptuel de données (MCD)**
   - Diagramme Entité-Association (ex : Film–Cinema M:N, Cinema–Room 1:N, Room–Screening 1:N, Screening–Reservation 1:N, User–Review 1:N, etc.).
5. **Diagrammes UML**
   - Cas d'utilisation principaux (Visiteur, Utilisateur, Employé, Admin),
   - Diagramme de séquence pour la réservation complète (du choix du film jusqu'à la génération du QR code).

6. **Plan de test**
    - Liste des tests unitaires (PHPUnit) pour entités (calcul note moyenne, places dispo, etc.),
    - Tests fonctionnels (pour contrôleurs sensibles) : vérifier accessibilité pages, formulaire login, formulaire contact, etc.
    - Tests de charge / performance (JMeter ou simili, si vous avez le temps).
7. **Déploiement continu (CI/CD)**
    - Présentation du workflow GitHub Actions,
    - Critères d'intégration (phpunit, phpstan, php-cs-fixer),
    - Dockerfile, docker-compose,
    - Procfile Heroku, variables, etc.

---

10. Résumé des commandes clés à lancer

1. **Sur la branche `refacto/symfony-ux`** :
2. `git checkout DEV`
3. `git pull origin DEV`
4. `git checkout -b refacto/symfony-ux`
5. **Installation et config Symfony** :
6. `composer require symfony/ux-twig-component symfony/ux-stimulus symfony/asset`
7. `rm -rf package.json node_modules webpack.config.js`
8. `mkdir -p assets/css assets/js/controllers assets/images`
9. `# Copier manuellement les CSS existants dans assets/css`
10. `# Créer un index Stimulus minimal dans assets/js/controllers/index.js`
11. **Migrations Doctrine** :
12. `php bin/console make:migration`
13. `php bin/console doctrine:migrations:migrate`
14. **Tests et qualité** :
15. `composer require --dev phpstan/phpstan phpstan/phpstan-doctrine friendsofphp/php-cs-fixer phpunit/phpunit symfony/phpunit-bridge`
16. `vendor/bin/phpstan analyse`
17. `vendor/bin/php-cs-fixer fix --diff --verbose`
18. `php bin/phpunit`
19. **Docker (en local)** :
20. `docker-compose up -d`
21. **Déploiement Heroku (après configuration)** :
22. `heroku git:remote -a cinephoria-app`
23. `git push heroku refacto/symfony-ux:main`
24. `heroku run php bin/console doctrine:migrations:migrate --no-interaction`

---

11. Conseils pour rendre le code "facile à expliquer" au jury

1. **Noms clairs et cohérents**
    - Entités, Repositories, Contrôleurs : utilisez la même terminologie que le cahier des charges (ex : `Screening` pour "séance", pas "Showtime").

    - Méthodes descriptives (`findLastWednesdayFilms()`, `recalcAverageRating()`, etc.).
2. **Annotations de type / PHPDoc**
    - Ajoutez systématiquement `@param` / `@return` quand le type n'est pas évident, même si PHP 8.3 le gère directement.
    - Cela facilite la relecture et la compréhension.
3. **Séparation claire des responsabilités**
    - **Entités** : seulement propriétés + getters/setters + méthodes métier basiques (recalcule note, décrémente places).
    - **Repositories** : uniquement requêtes personnalisées.
    - **Services** : si vous avez de la logique complexe (ex : génération de QR code, envoi de mails), déplacez-la dans un service dédié (ex : `src/Service/QRCodeGenerator.php`).
    - **Contrôleurs** : doivent être fins, ne pas contenir beaucoup de logique métier. S'ils commencent à grossir, extrayez des services (ex : `ReservationManager`, `ReviewValidator`, etc.).
4. **FormTypes**
    - Utilisez les FormTypes pour tout formulaire, même si c'est simple, plutôt que de gérer les `$_POST`manuellement.
    - Ajoutez dans chaque FormType des contraintes (`Assert`) pour valider côté serveur.
5. **Commentaires ponctuels**
    - Là où la logique peut être subtile (ex : calcul places disponibles, pipeline MongoDB pour stats), ajoutez un commentaire court expliquant "pourquoi" ce code plutôt que "comment".
6. **Organisation des templates**
    - Chaque template doit être découpé en blocs Twig (`base.html.twig`, `navbar.html.twig`, `footer.html.twig`) et inclus par les pages, de façon à ce que le jury voit clairement la structure.
7. **Cohérence des routes**
    - Privilégiez des noms de route explicites (`film_list`, `admin_film_new`, `employee_incident_list`) et des URL qui reflètent l'arborescence (`/films`, `/admin/film/new`, `/employee/incidents`).
8. **Fichier de configuration unique**
    - Ne dupliquez pas la configuration de sécurité, placez tout dans `security.yaml` avec des commentaires précisant "ROLE_USER = client", "ROLE_EMPLOYEE = employé", "ROLE_ADMIN = administrateur".
9. **Tests significatifs**
    - Fournissez au moins un test unitaire pour chaque entité critique (Film, Screening, Reservation),
    - Un test fonctionnel pour chaque type d'utilisateur (visiteur, utilisateur, employé, admin).

En synthèse, vous devrez :

1. **Basculer** sur `DEV`, créer `refacto/symfony-ux`.

2. **Supprimer** toute dépendance à ReactJS/Node/Webpack ; installer Symfony UX (Stimulus) + Asset Mapper.
3. **Refondre** toutes les entités Doctrine (User, Cinema, Room, Film, Screening, Reservation, Review, Incident).
4. **Mettre à jour** tous les **Repositories** avec les requêtes métier (dernier mercredi, filtrages, etc.).
5. **Reprendre** tous les **Contrôleurs** (public, security, utilisateur, employé, admin) pour qu'ils rendent des pages Twig.
6. **Recréer** tous les **templates Twig** (base.html.twig, index, film_list, film_show, reservation, user_dashboard, admin_*, employee_*).
7. **Intégrer** Stimulus (Symfony UX) pour les comportements légers (chargement dynamique des séances, calcul du prix).
8. **Configurer** PHPStan, PHP-CS-Fixer, PHPUnit et corriger les erreurs.
9. **Dockeriser** l'application avec un `Dockerfile` et un `docker-compose.yml`.
10. **Configurer** une pipeline GitHub Actions pour tests, analyse et style.
11. **Préparer** le déploiement sur Heroku (Procfile, config var).
12. **Documenter** : README, manuel d'utilisation, charte graphique, gestion de projet, documentation technique.

Conclusion
En suivant cette feuille de route exhaustive, vous transformerez votre projet initial "Cinephoria" en une application Symfony monolithique, claire, maintenable et 100 % PHP/Twig/Stimulus :

- **Le front** est désormais géré en Twig + Stimulus (Symfony UX) sans React,
- **Le back** est structuré en entités bien typées, contrôleurs fins, repositories dédiés,
- **La qualité** est assurée par PHPStan, PHPUnit et PHP-CS-Fixer,
- **Le déploiement** se fait via Docker + GitHub Actions + Heroku (ou VPS),
- **Le code** sera facile à expliquer au jury grâce à une nomenclature limpide, une séparation nette des responsabilités et une documentation détaillée.

N'hésitez pas, après avoir terminé chaque section, à relire vos modifications avec un œil de pair ou un formateur, pour vous assurer que tout est clair et que vous pourrez l'expliquer aisément le jour de l'oral. Bonne refonte !