* La dépendance devtools permet d’actualiser la page sans relancer l’execution du projet, pour l’activer :
  + Advanced settings -> Cocher allow auto-make to start even if developed application is currently running.
  + Build, execution, deployment -> Cocher build project automatically.
* List<Patient> patientList = patientRepository.findAll();
* @GetMapping("/index")  
  public String index(Model model,  
   @RequestParam(name="page",defaultValue="0") int page,  
   @RequestParam(name="size",defaultValue="4") int size,  
   @RequestParam(name="keyword",defaultValue="") String keyword){  
   Page<Patient> pagePatients = patientRepository.findByNomContains(keyword, PageRequest.*of*(page,size));  
   model.addAttribute("listpatients",pagePatients.getContent());  
   model.addAttribute("pages", new int[pagePatients.getTotalPages()]);  
   model.addAttribute("currentPage",page);  
   model.addAttribute("keyword",keyword);  
   return "patients";  
  }
* Pour eviter l’erreur l’hors de l’enregistrement d’un nouveau patient, il faut indiquer le format de la date dans application.properties
* spring.mvc.format.date=yyyy-MM-dd
* pour les ajouter les annotations de validations il faut ajouter la dépendance de spring boot validation
* Avec une requête https le mot de passe envoyé est dans un canal sécurisé, contrairement à si le mot de passe est envoyé via http tout le monde peux l’intercepter
* <dependency>  
   <groupId>org.thymeleaf.extras</groupId>  
   <artifactId>thymeleaf-extras-springsecurity6</artifactId>  
   <version>3.1.0.M1</version>  
  </dependency>
* Methode 2 de securisation:
  + Ajouter dans le controleur avant chaque requette

@PreAuthorize("hasRole('ADMIN')")

* + Ajouter dans SecurityConfig
* @EnableMethodSecurity(prePostEnabled = true)
* Securisation Front: cacher l’accès visuel vers les requêtes html, les requêtes restent accessible via l’url

th:if="${#authorizati

on.expression('hasRole(''ADMIN'')')}"

# JDBC Authentication

* Authentification statefull: les données de la session sont enregistrées coté serveur d’authentification ( après validation des identifiants, un ID de la session est crée et stocké dans les coockies
* Authentification stateless : suite à l’authentification un token est généré (parmi les token connu json web token) , le token ensuite est envoyé vers le client et stocké chez lui en local storage
* Le stockage en authentification statefull crée une faille de sécurité appelée Cross site request forgery . Pour l’eliminer, il faut que à chaque fois un formulaire est demandé au serveur, un token est généré et placé dans la session, ensuite le serveur renvoie le formulaire avec un champ caché contenant ce token CSRF, ensuite quand on fait une nouvelle requête on peut vérifier le tocken caché envoyé dans le formulaire est le même que celui qui stocké dans la session (coté serveur)

De ce fait, je suis sur que le formulaire envoyé provient de moi et qu’il ne s’agit pas d’une attaque CSRF

# User details service

# Authentication via Gmail, FB ou autre