DÉVELOPPEMENT BACKEND

Spring Boot : développer des API REST en Java

Créer des API REST robustes et sécurisées avec Spring Boot. Injection de dépendances, Spring Data JPA, Spring Security avec JWT, gestion des erreurs et déploiement : le standard du backend Java en entreprise.

DURÉE	TARIF HT	NIVEAU	LANGUE	GROUPE	FORMAT
5j (35h)	**1390.00 €**	**Intermédiaire**	**Français**	**3-12**	**Formation**

1 PUBLIC VISÉ

- Développeurs Java maîtrisant la POO souhaitant créer des API REST professionnelles avec Spring Boot.
- Personnes ayant suivi la formation Les fondamentaux de la programmation Java et POO (BCK-JAV) ou disposant d'une expérience Java équivalente.
- Développeurs backend souhaitant adopter le framework le plus utilisé dans les environnements Java enterprise.
- Personnes en reconversion visant un poste de développeur Java backend ou full-stack.

2 PRÉREQUIS

- Bonne maîtrise de Java et de la programmation orientée objet : classes, héritage, interfaces, collections, exceptions, streams.
- La formation Les fondamentaux de la programmation Java et POO est requise ou une expérience Java équivalente.
- Notions de SQL et de base de données relationnelle.
- Savoir utiliser le terminal et IntelliJ IDEA.

3 OBJECTIFS PÉDAGOGIQUES

- Comprendre l'architecture de Spring Boot et le mécanisme d'auto-configuration
- Maîtriser l'injection de dépendances et l'inversion de contrôle avec le conteneur Spring
- Créer des endpoints REST avec Spring MVC : GET, POST, PUT, PATCH, DELETE
- Concevoir et utiliser des DTOs pour découpler l'API de la couche persistance
- Persister des données avec Spring Data JPA et Hibernate
- Gérer les relations entre entités : OneToMany, ManyToOne, ManyToMany
- Sécuriser une API REST avec Spring Security et l'authentification par JWT
- Gérer les erreurs et exceptions de manière globale avec un handler centralisé
- Tester une API REST avec JUnit 5 et MockMvc
- Déployer une application Spring Boot en JAR exécutable ou via Docker

4 PROGRAMME DÉTAILLÉ

Jour 1 (7h) – Architecture Spring Boot et premiers endpoints

Module 1 – Découverte de Spring Boot (2h)

Pourquoi Spring Boot ? (45min)

- Spring Framework vs Spring Boot : ce que Boot simplifie concrètement
- L'auto-configuration : comment Spring Boot devine ce dont vous avez besoin
- Spring Initializr : générer un projet en quelques clics
- Les starters Spring Boot : spring-boot-starter-web, -data-jpa, -security
- Spring Boot dans l'écosystème Java enterprise : Netflix, Airbnb, Zalando

Structure et démarrage du projet (1h15)

- Créer un projet avec Spring Initializr : Java 21, Maven, dépendances
- Structure du projet : src/main/java, src/main/resources, pom.xml
- La classe principale annotée @SpringBootApplication
- Le fichier application.properties et application.yml
- Lancer l'application et lire les logs de démarrage
- Spring Boot DevTools : rechargement automatique en développement
- Cas pratique : créer et démarrer un projet Spring Boot vide

Module 2 – Injection de dépendances et conteneur Spring (2h)

L'inversion de contrôle (1h)

- Le problème que l'IoC résout : couplage fort et testabilité
- Le conteneur Spring ApplicationContext : créer et gérer les beans
- Les annotations de stéréotype : @Component, @Service, @Repository, @Controller
- @Autowired et l'injection par constructeur : la méthode recommandée
- @Bean dans une classe @Configuration : déclarer un bean manuellement
- Cas pratique : service métier injecté dans un composant via le constructeur

Portée et cycle de vie des beans (1h)

- La portée Singleton : un seul bean par conteneur par défaut
- Les autres portées : Prototype, Request, Session
- @PostConstruct et @PreDestroy : hooks du cycle de vie
- Les profils Spring : @Profile pour différencier dev, test, prod
- Cas pratique : configurer des beans différents selon le profil actif

Module 3 – Contrôleurs REST et DTOs (3h)

Créer des endpoints REST (1h30)

- @RestController et @RequestMapping : déclarer un contrôleur REST
- Les annotations de mapping : @GetMapping, @PostMapping, @PutMapping, @PatchMapping, @DeleteMapping
- @PathVariable : récupérer un paramètre dans l'URL
- @RequestParam : récupérer un paramètre de requête
- @RequestBody : désérialiser le corps JSON de la requête
- ResponseEntity : contrôler le statut HTTP et les headers de la réponse
- Cas pratique : CRUD complet sur une ressource avec les bons statuts HTTP

DTOs et validation (1h30)

- Pourquoi les DTOs : ne jamais exposer les entités JPA directement
- DTO de requête (CreateRequest, UpdateRequest) et de réponse (Response)
- Mapper manuellement entre entité et DTO
- La validation avec Bean Validation : @NotNull, @NotBlank, @Size, @Email, @Min, @Max
- @Valid sur le @RequestBody : déclencher la validation automatiquement
- Cas pratique : ressource Utilisateur avec DTOs distincts pour la création et la réponse

Jour 2 (7h) – Spring Data JPA et persistance

Module 4 – Entités JPA et Spring Data (3h)

Modéliser des entités JPA (1h)

- @Entity, @Table, @Id, @GeneratedValue : les annotations de base
- Les types de colonnes : @Column, @Enumerated, @Temporal
- @CreatedDate et @LastModifiedDate avec Spring Data Auditing
- Configurer la source de données dans application.yml
- Générer le schéma avec spring.jpa.hibernate.ddl-auto
- Cas pratique : entités Article et Auteur avec auditing automatique

Repositories Spring Data JPA (2h)

- JpaRepository : les méthodes CRUD offertes gratuitement
- Les méthodes dérivées : findByNom, findByEmailAndActif, countByCategorie
- @Query avec JPQL : écrire des requêtes orientées objet
- @Query avec SQL natif : nativeQuery = true pour les cas complexes
- Pagination et tri avec Pageable et Page
- Projections : retourner uniquement les champs nécessaires
- Cas pratique : repository Article avec recherche par mot-clé et pagination

Module 5 – Relations entre entités (2h)

Mapper les relations (1h30)

- @ManyToOne et @OneToMany : relation bidirectionnelle article/catégorie
- @ManyToMany : relation article/tag avec @JoinTable
- @OneToOne : relation utilisateur/profil
- Le côté propriétaire et mappedBy : éviter les boucles infinies en JSON
- CascadeType : persist, merge, remove — ce que chaque valeur implique
- orphanRemoval : supprimer les enfants orphelins automatiquement

Optimiser le chargement des relations (30min)

- FetchType.LAZY vs EAGER : le problème N+1 expliqué
- Résoudre le N+1 avec les JOIN FETCH dans JPQL
- @EntityGraph : charger les relations de manière sélective
- Cas pratique : API d'articles avec catégories et tags sans N+1

Module 6 – Couche service et transaction (2h)

La couche service (1h)

- Pourquoi séparer Controller, Service et Repository

@Service : déclarer la couche métier

- Règles de gestion dans le service : validation métier, orchestration
- Lever des exceptions métier depuis le service
- Cas pratique : service ArticleService avec règles métier

Gestion des transactions (1h)

- @Transactional : garantir l'atomicité d'une opération
- Propagation des transactions : REQUIRED, REQUIRES_NEW
- readOnly = true : optimiser les transactions de lecture
- Rollback automatique sur les RuntimeException
- Cas pratique : service de transfert avec rollback en cas d'erreur

Jour 3 (7h) – Sécurité avec Spring Security et JWT

Module 7 – Spring Security : authentification JWT (4h)

Comprendre Spring Security (1h)

- L'architecture de Spring Security : filtres, AuthenticationManager, SecurityContext
- Configurer la sécurité avec SecurityFilterChain
- Autoriser et restreindre les routes avec requestMatchers
- CORS : configurer les origines autorisées pour une API consommée par un front-end
- CSRF : pourquoi le désactiver pour une API REST stateless

Authentification par JWT (2h)

- Qu'est-ce qu'un JWT : header, payload, signature
- Générer un JWT à la connexion avec la bibliothèque jjwt
- Créer le filtre JwtAuthenticationFilter pour valider le token à chaque requête
- Stocker l'utilisateur authentifié dans le SecurityContext
- Endpoints publics et endpoints protégés : la configuration complète
- Refresh tokens : renouveler l'accès sans redemander les identifiants
- Cas pratique : endpoints /auth/register et /auth/login avec génération de JWT

Autorisation et rôles (1h)

- Les rôles et authorities dans Spring Security
- @PreAuthorize : protéger une méthode par rôle ou expression SpEL
- hasRole, hasAuthority, @CurrentSecurityContext : récupérer l'utilisateur connecté
- Cas pratique : endpoints réservés aux administrateurs avec @PreAuthorize

Module 8 – Gestion des erreurs globale (3h)

Centraliser la gestion des erreurs (1h30)

- Le problème des erreurs non gérées : stack traces exposées en production
- @ControllerAdvice et @ExceptionHandler : intercepter toutes les exceptions
- Créer un format de réponse d'erreur standardisé : ErrorResponse
- Gérer les erreurs de validation Bean Validation : MethodArgumentNotValidException
- Gérer les 404 avec NoSuchElementException et ResourceNotFoundException
- Cas pratique : handler global avec réponses d'erreur JSON cohérentes

Logging et monitoring (1h30)

SLF4J et Logback : le standard de logging Java avec Spring Boot

- Les niveaux de log : ERROR, WARN, INFO, DEBUG, TRACE
- Configurer les niveaux par package dans application.yml
- Spring Boot Actuator : endpoints de monitoring /health, /info, /metrics
- Cas pratique : ajouter des logs structurés aux opérations critiques de l'API

Jour 4 (7h) – Tests et documentation

Module 9 – Tests avec JUnit 5 et MockMvc (4h)

Tests unitaires avec JUnit 5 (1h30)

- Structure d'un test JUnit 5 : @Test, @BeforeEach, @AfterEach
- Les assertions : assertEquals, assertThrows, assertThat avec AssertJ
- Tester un service avec Mockito : @Mock, @InjectMocks, when().thenReturn()
- Vérifier les interactions avec verify()
- Cas pratique : tests unitaires du service ArticleService avec mocks du repository

Tests d'intégration avec MockMvc (1h30)

- @SpringBootTest : charger le contexte complet pour les tests d'intégration
- @WebMvcTest : tester la couche web isolément
- MockMvc : simuler des requêtes HTTP et vérifier les réponses
- @WithMockUser : tester les endpoints sécurisés
- Cas pratique : tests d'intégration des endpoints CRUD avec vérification des statuts et du JSON

Tests de la couche repository (1h)

- @DataJpaTest : tester les repositories avec une base H2 en mémoire
- Tester les méthodes dérivées et les requêtes JPQL personnalisées
- Cas pratique : tests du repository Article avec données de test

Module 10 – Documentation avec Swagger et OpenAPI (3h)

Documenter une API REST (1h30)

- Pourquoi documenter une API : contrat entre front-end et back-end
- Intégrer springdoc-openapi dans le projet
- L'interface Swagger UI : explorer et tester l'API depuis le navigateur
- @Operation, @ApiResponse, @Parameter : enrichir la documentation
- @Schema : documenter les DTOs
- Configurer l'authentification JWT dans Swagger UI
- Cas pratique : documenter tous les endpoints de l'API avec Swagger

Versioning et bonnes pratiques API REST (1h30)

- Les conventions REST : nommage des ressources, statuts HTTP appropriés
- Versioning d'API : URI versioning, header versioning
- HATEOAS : enrichir les réponses avec des liens de navigation
- Rate limiting : protéger l'API contre les abus
- Cas pratique : refactoring de l'API pour respecter les conventions REST

Jour 5 (7h) – Déploiement et projet de synthèse

Module 11 – Déploiement (3h)

Packaging et configuration production (1h)

- Construire le JAR exécutable avec Maven : mvn clean package
- Les profils Spring : application-dev.yml, application-prod.yml
- Les variables d'environnement : externaliser les secrets hors du code
- Configurer une base de données PostgreSQL en production
- Flyway : gérer les migrations de base de données en production

Conteneurisation avec Docker (2h)

- Écrire un Dockerfile pour une application Spring Boot
- Optimiser l'image Docker : multi-stage build, image JRE légère
- Docker Compose : orchestrer l'application et la base de données
- Variables d'environnement dans Docker Compose
- Cas pratique : conteneuriser l'API avec Docker Compose et PostgreSQL

Module 12 – Projet de synthèse (4h)

Réalisation d'une API REST Spring Boot complète

- Cahier des charges : API de gestion de bibliothèque ou de plateforme de blog
- Fonctionnalités : architecture en couches Controller/Service/Repository, entités JPA avec relations, Spring Data JPA avec pagination et filtres, authentification JWT avec rôles ADMIN et USER, DTOs distincts pour les requêtes et les réponses, gestion des erreurs globale avec ErrorResponse, tests unitaires sur les services critiques, documentation Swagger complète, déploiement avec Docker Compose
- Étape 1 : modélisation des entités et configuration JPA
- Étape 2 : endpoints CRUD avec DTOs et validation
- Étape 3 : sécurité JWT et gestion des rôles
- Étape 4 : tests, documentation et Docker
- Revue de code collective : architecture, sécurité, qualité des tests
- Retour formateur individualisé sur le projet rendu

5 COMPÉTENCES VISÉES

- Concevoir et développer une API REST complète et documentée avec Spring Boot
- Modéliser et interroger une base de données relationnelle avec Spring Data JPA
- Sécuriser une API avec Spring Security et des tokens JWT
- Écrire des tests unitaires et d'intégration pour valider le comportement des endpoints
- Structurer un projet Spring Boot selon les bonnes pratiques : couches Controller, Service, Repository
- Déployer une application Spring Boot en production via JAR ou conteneur Docker

6 MODALITÉS PÉDAGOGIQUES

Formation délivrée en présentiel ou distanciel (visioconférence). Le formateur alterne entre méthode démonstrative (live coding commenté sur une API construite progressivement tout au long de la formation), méthode interrogative (analyse des choix d'architecture et des conventions REST) et méthode active (exercices pratiques et mini-projets par module). L'accent est mis sur les pratiques professionnelles réelles : séparation des couches, sécurité, tests et déploiement.

7 MOYENS PÉDAGOGIQUES ET TECHNIQUES

- Support de cours numérique mis à disposition des apprenants
- Dépôt GitHub de démonstration avec exercices et corrections par module
- Environnement de développement : IntelliJ IDEA + JDK 21 + Docker Desktop
- Pour le distanciel : visioconférence (Zoom ou équivalent), partage d'écran, chat en direct
- Accès à la plateforme pédagogique LaPolaris (supports, ressources, émargement)

8 MODALITÉS D'ÉVALUATION

- En cours de formation : exercices pratiques et mini-projets corrigés à chaque module
- En fin de formation : réalisation d'une API REST Spring Boot complète avec authentification JWT, tests et documentation
- Questionnaire d'auto-évaluation des acquis en fin de parcours

9 CRITÈRES D'ÉVALUATION

- Architecture en couches respectée : Controller, Service, Repository sans mélange des responsabilités
- Endpoints REST conformes aux conventions : nommage, statuts HTTP, DTOs distincts
- Authentification JWT fonctionnelle avec protection des routes selon les rôles
- Tests unitaires couvrant les cas nominaux et les cas d'erreur des services critiques
- Gestion des erreurs centralisée avec des réponses JSON cohérentes et sans stack trace exposée

10 MODALITÉS DE VALIDATION

Attestation de fin de formation délivrée à l'issue du parcours, conditionnée à une assiduité d'au moins 80 % et à la réalisation du projet de synthèse. L'attestation précise les objectifs atteints et les compétences acquises.

11 SUIVI ET ACCOMPAGNEMENT

- Feuilles d'émargement signées par demi-journée (présentiel) ou émargement numérique (distanciel)
- Traçabilité des activités pédagogiques réalisées
- Attestation d'assiduité délivrée en fin de formation
- Suivi individuel via les exercices corrigés et le projet de synthèse

12 CONDITIONS D'ACCÈS

Formation accessible sur inscription directe, sans prérequis administratif particulier. Le financement peut être pris en charge par l'employeur dans le cadre d'un plan de développement des compétences, ou en autofinancement avec possibilité de paiement en plusieurs fois.

13 DÉLAIS D'ACCÈS

Inscription possible jusqu'à **5 jours ouvrés** avant le début de la session. Pour toute demande urgente, nous contacter directement.

ACCESSIBILITÉ · HANDICAP

Nos formations sont accessibles aux personnes en situation de handicap. Pour toute situation nécessitant un aménagement (matériel, temporel ou pédagogique), nous vous invitons à nous contacter avant l'inscription afin d'étudier les adaptations possibles. Référent handicap : contact@lapolaris.fr

LaPolaris

TÉL. +33762584798

EMAIL contact@lapolaris.fr

WEB lapolaris.fr