Spring Boot — Cheatsheet Ultra Complet (Version Avancée)

Table des matières

Principales annotations

Annotation	Utilité	Exemple
`@SpringBootApplication`	Point d'entrée principal	`@SpringBootApplication`
`@RestController`	Contrôleur REST (retourne JSON/XML)	`@RestController`
`@Controller`	Contrôleur MVC classique	`@Controller`
`@Service`	Service métier	`@Service`
`@Repository`	DAO, accès aux données	`@Repository`
`@Autowired`	Injection automatique	`@Autowired`
`@Qualifier`	Choix du bean injecté si plusieurs disponibles	`@Qualifier("monBean")`
`@Value`	Injection de valeur (properties, env)	`@Value("${app.name}")`
`@Component`	Composant générique	`@Component`
`@Configuration`	Classe de config	`@Configuration`
`@Bean`	Déclare un bean	`@Bean`
`@GetMapping`, `@PostMapping`	Mapping HTTP (GET, POST)	`@GetMapping("/users")`

Annotation	Utilité	Exemple
`@PutMapping`, `@DeleteMapping`	Mapping HTTP (PUT, DELETE)	`@DeleteMapping("/users/{id}")`
`@PathVariable`	Paramètre d'URL	`@GetMapping("/user/{id}")`
`@RequestParam`	Paramètre de requête	`@RequestParam("page")`
`@RequestBody`	Corps de la requête HTTP	`@RequestBody User user`
`@ModelAttribute`	Pour lier un objet de formulaire	`@ModelAttribute User user`
`@Entity`	Objet persistant JPA	`@Entity`
`@Table`	Nom de table JPA	`@Table(name="users")`
`@Id`	Clé primaire	`@Id`
`@GeneratedValue`	Génération automatique de l'ID	`@GeneratedValue(strategy=...)`
`@Column`	Spécifie une colonne JPA	`@Column(nullable=false)`
`@Transactional`	Gestion de transaction	`@Transactional`
`@ExceptionHandler`	Gestion des exceptions contrôleur	`@ExceptionHandler(Exception.class)`
`@ControllerAdvice`	Gestion globale des exceptions	`@ControllerAdvice`
`@Profile`	Bean pour un profil spécifique	`@Profile("dev")`
`@PreAuthorize`	Sécurité par méthode (Spring Security)	`@PreAuthorize("hasRole('ADMIN')")`
`@Scheduled`	Planification de tâches	`@Scheduled(cron = "0 0 * * * *")`
`@Cacheable`	Mise en cache automatique	`@Cacheable("users")`

Avec Lombok :
Réduit le code répétitif, utile pour les modèles et DTOs :

- `@Data`, `@Getter`, `@Setter`, `@NoArgsConstructor`, `@AllArgsConstructor`, `@Builder`, `@Slf4j`, `@ToString`, `@EqualsAndHashCode`.

Astuces pratiques :

- Combine plusieurs annotations avec `@SpringBootTest(properties = {"spring.profiles.active=test"})`

- Pour les tests, `@MockBean` (Spring) ou `@Mock` (Mockito) pour injecter des mocks.
- Utilise `@JsonIgnore` (Jackson) pour masquer certains champs à la sérialisation.

Architecture MVC avec Spring Boot

Architecture recommandée

```
src/main/java/
 ├─ controller/   (web, REST)
 ├─ service/      (business)
 ├─ repository/   (JPA/DAO)
 └─ model/        (entités/dto)
```

Modèle — Validation avancée

```java
@Entity
public class User {
    @Id @GeneratedValue
    private Long id;

    @NotBlank(message="Le nom est obligatoire")
    @Size(min=2, max=50)
    private String name;

    @Email(message="Email invalide")
    @Column(unique=true)
    private String email;

    @Past
    private LocalDate birthDate;

    @Enumerated(EnumType.STRING)
    private Role role;

    // Lombok @Data, @NoArgsConstructor, etc.
}
```

Astuces validation :

- Utilise aussi `@Pattern`, `@Max`, `@Min`, `@AssertTrue`…
- Pour la validation côté service, utilise `@Validated` sur le service.

Repository — requêtes personnalisées

```java
public interface UserRepository extends JpaRepository<User, Long> {
    List<User> findByRole(Role role);
```

```
    // Requête JPQL personnalisée
    @Query("SELECT u FROM User u WHERE u.name LIKE %:name%")
    List<User> searchByName(@Param("name") String name);

    // Projection DTO
    @Query("SELECT new com.example.demo.dto.UserNameEmail(u.name, u.email)
FROM User u")
    List<UserNameEmail> findNameAndEmail();
}
```

Service — transaction, logique métier

```
@Service
public class UserService {
    @Autowired UserRepository repo;

    @Transactional(readOnly = true)
    public List<User> getUsersByRole(Role role) { return
repo.findByRole(role); }

    @Transactional
    public User createUser(User user) {
        if (repo.existsByEmail(user.getEmail())) {
            throw new BusinessException("Email déjà utilisé");
        }
        // Traitement métier...
        return repo.save(user);
    }

    // Exemple d'appel à un autre service
    @Autowired private MailService mailService;

    public void notifyUser(User user) {
        mailService.sendWelcome(user.getEmail());
    }
}
```

Contrôleur — MVC avancé

```
@Controller
@RequestMapping("/users")
public class UserController {
    @Autowired UserService userService;

    @GetMapping
    public String list(@RequestParam Optional<String> q, Model model) {
        model.addAttribute("users",
```

```
q.map(userService::searchByName).orElseGet(userService::findAll));
        return "users/list";
    }

    @GetMapping("/edit/{id}")
    public String edit(@PathVariable Long id, Model model) {
        model.addAttribute("user", userService.getById(id));
        return "users/form";
    }

    @PostMapping
    public String save(@Valid @ModelAttribute User user, BindingResult
result, Model model) {
        if (result.hasErrors()) return "users/form";
        try {
            userService.createUser(user);
        } catch (BusinessException ex) {
            model.addAttribute("error", ex.getMessage());
            return "users/form";
        }
        return "redirect:/users";
    }
}
```

Astuces :

- Ajoute `@ModelAttribute("roles")` pour injecter une liste dans tous les formulaires.
- Utilise `Principal` ou `@AuthenticationPrincipal` pour obtenir l'utilisateur connecté.

Gestion des formulaires avec Thymeleaf

Bonnes pratiques

- Utilise `th:object` pour binder un bean.
- `th:field` pour lier chaque champ.
- Affiche les erreurs de validation avec `th:errors` et `${#fields.hasErrors('property')}`.
- Pour les listes (`select`, `checkboxes`) : injecte les valeurs via le modèle.

Formulaire utilisateur complet

```
<form th:action="@{/users}" th:object="${user}" method="post">
    <input type="text" th:field="*{name}" placeholder="Nom" class="form-
control"/>
    <div th:if="${#fields.hasErrors('name')}" class="text-danger"
th:errors="*{name}"></div>
    <input type="email" th:field="*{email}" placeholder="Email"
class="form-control"/>
    <div th:if="${#fields.hasErrors('email')}" class="text-danger"
th:errors="*{email}"></div>
    <select th:field="*{role}" class="form-control">
```

```
        <option th:each="r : ${roles}" th:value="${r}" th:text="${r}">
</option>
    </select>
    <input type="date" th:field="*{birthDate}" class="form-control"/>
    <button type="submit" class="btn btn-primary">Valider</button>
    <div th:if="${error}" class="alert alert-danger" th:text="${error}">
</div>
</form>
```

Traitement d'upload de fichiers

Contrôleur :

```
@PostMapping(consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
public String handleUpload(@RequestParam MultipartFile file, Model model)
throws IOException {
    if (file.isEmpty()) {
        model.addAttribute("error", "Fichier manquant");
        return "users/form";
    }
    // Sauvegarde ou traitement...
    return "redirect:/users";
}
```

Formulaire Thymeleaf :

```
<form th:action="@{/users/upload}" method="post" enctype="multipart/form-
data">
    <input type="file" name="file" class="form-control"/>
    <button type="submit">Upload</button>
</form>
```

Astuces avancées

- Pour afficher des messages flash :
  Utilise `RedirectAttributes` et `${flash.message}`.
- Pour la pagination dans la vue :
  Boucle sur les pages et construis les liens avec `th:href="@{/users(page=${i})}"`.
- Pour internationaliser les messages d'erreur :
  Place les messages dans `messages.properties` et utilise `th:text="#{error.field}"`.

Création et utilisation de REST APIs

Contrôleur REST ultra-complet

```java
@RestController
@RequestMapping("/api/users")
@CrossOrigin(origins = "http://localhost:4200") // précis ou *
public class UserRestController {
    @Autowired UserService userService;

    @GetMapping
    public Page<User> getAll(@RequestParam(defaultValue="0") int page,
@RequestParam(defaultValue="10") int size) {
        return userService.getPage(page, size);
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> getById(@PathVariable Long id) {
        return userService.findById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<?> create(@RequestBody @Valid User user) {
        try {
            return
ResponseEntity.status(HttpStatus.CREATED).body(userService.createUser(user)
);
        } catch (BusinessException ex) {
            return
ResponseEntity.status(HttpStatus.CONFLICT).body(Map.of("error",
ex.getMessage()));
        }
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> update(@PathVariable Long id, @RequestBody
@Valid User user) {
        try {
            return ResponseEntity.ok(userService.updateUser(id, user));
        } catch (EntityNotFoundException e) {
            return ResponseEntity.notFound().build();
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> delete(@PathVariable Long id) {
        if (!userService.existsById(id)) return
ResponseEntity.notFound().build();
        userService.delete(id);
        return ResponseEntity.noContent().build();
    }
}
```

Gestion fine des erreurs

```
@ControllerAdvice
public class RestExceptionHandler {
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, String>>
handleValidation(MethodArgumentNotValidException ex) {
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getFieldErrors().forEach(err ->
errors.put(err.getField(), err.getDefaultMessage()));
        return ResponseEntity.badRequest().body(errors);
    }

    @ExceptionHandler(BusinessException.class)
    public ResponseEntity<?> handleBusiness(BusinessException ex) {
        return
ResponseEntity.status(HttpStatus.CONFLICT).body(Map.of("error",
ex.getMessage()));
    }
}
```

Pagination & tri REST

- Utilise Spring Data :
  `Page<User> findAll(Pageable pageable)`
  Appel : `/api/users?page=2&size=20&sort=name,desc`

Exposition sélective des données

- Utilise DTOs pour ne jamais exposer d'entité brute.
- Ou, pour masquer des champs :
  `@JsonIgnore`, `@JsonView`, ou projections Spring Data.
- Pour les dates, ajoute dans `application.properties` :
  `spring.jackson.date-format=yyyy-MM-dd`

Sécuriser l'API

- Limite les méthodes HTTP exposées.
- Ajoute des contrôles d'accès :
  `@PreAuthorize("hasRole('ADMIN')")` sur les méthodes sensibles.

Documentation automatique

- Ajoute `springdoc-openapi-ui` ou `springfox-swagger-ui` :
  - Swagger UI : `/swagger-ui.html`
  - OpenAPI JSON : `/v3/api-docs`

Astuces avancées :

- Pour les uploads REST :
  `@PostMapping(consumes=MediaType.MULTIPART_FORM_DATA_VALUE)` et `@RequestParam`

`MultipartFile file`

- Pour la pagination REST : retourne l'objet Page et pas seulement le contenu, pour exposer le nombre de pages, etc.

Sécurité avec Spring Security

Configuration moderne (Spring Security 6+)

```
@Configuration
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
        return http
            .authorizeHttpRequests(authz -> authz
                .requestMatchers("/", "/home", "/css/**",
"/api/public/**").permitAll()
                .requestMatchers("/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated())
            .formLogin(form -> form
                .loginPage("/login").permitAll())
            .logout(logout -> logout.permitAll())
            .csrf(csrf -> csrf.disable()) // réactiver en prod
            .build();
    }

    @Bean
    public UserDetailsService users() {
        UserDetails user =
User.withDefaultPasswordEncoder().username("user").password("password").rol
es("USER").build();
        UserDetails admin =
User.withDefaultPasswordEncoder().username("admin").password("admin").roles
("ADMIN").build();
        return new InMemoryUserDetailsManager(user, admin);
    }
}
```

Authentification via BDD

```
@Service
public class CustomUserDetailsService implements UserDetailsService {
    @Autowired UserRepository repo;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = repo.findByEmail(username).orElseThrow(() -> new
UsernameNotFoundException("Not found"));
```

```
        return new org.springframework.security.core.userdetails.User(
            user.getEmail(), user.getPassword(), List.of(new
SimpleGrantedAuthority("ROLE_" + user.getRole())));
    }
}
```

Sécurité des endpoints REST (JWT, OAuth2)

- Utilise la dépendance `spring-boot-starter-oauth2-resource-server`.
- Pour du JWT “stateless” : configure un filtre JWT ou la ressource server.

Exemple d’activation CORS global :

```
@Bean
public WebMvcConfigurer corsConfigurer() {
    return new WebMvcConfigurer() {
        @Override
        public void addCorsMappings(CorsRegistry registry) {
            registry.addMapping("/api/**").allowedOrigins("*");
        }
    };
}
```

Astuces sécurité

- Toujours stocker les mots de passe hashés :
  `@Bean PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }`
- Ajoute un CSRF token dans les formulaires :
  `<input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}"/>`
- Pour mocker un utilisateur de test :
  `@WithMockUser(roles = "ADMIN")` dans les tests.

Configuration avancée

Fichier `application.properties` ou `application.yml`

Exemples avancés :

```
# Serveur
server.port=8082
server.servlet.context-path=/myapp

# BDD
spring.datasource.url=jdbc:mysql://localhost:3306/demo
spring.datasource.username=root
```

```
spring.datasource.password=pass
spring.jpa.hibernate.ddl-auto=update

# Logs
logging.level.root=WARN
logging.level.org.springframework=INFO
logging.level.com.example.demo=DEBUG
logging.pattern.console=%d{HH:mm:ss} %-5level %logger{36} - %msg%n

# Thymeleaf
spring.thymeleaf.cache=false

# Swagger
springdoc.api-docs.path=/api-docs

# Multi-profil
spring.profiles.active=dev

# CORS global
spring.web.cors.allowed-origins=*

# Variables custom
app.name=Demo Ultra
app.admin.email=admin@demo.com
```

Utilisations avancées :

- Plusieurs fichiers : `application-dev.properties`, `application-prod.properties`
- Pour charger une variable d'environnement :
  `@Value("${MY_ENV_VAR:defaultValue}")`
- Pour définir des listes :
  `app.roles=ADMIN,USER,MANAGER`

Accès au contenu du fichier dans le code :

```
@Value("${app.admin.email}")
private String adminEmail;

@ConfigurationProperties(prefix = "app")
public class AppProperties {
    private String name;
    private List<String> roles;
    // getters/setters
}
```

Profil dynamique au lancement :

```
java -jar monapp.jar --spring.profiles.active=prod
```

Tests unitaires et d’intégration

Dépendances principales

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<!-- MockMvc, JUnit 5, Mockito, AssertJ inclus -->
```

Test de contrôleur MVC

```
@SpringBootTest
@AutoConfigureMockMvc
public class UserControllerTest {
    @Autowired MockMvc mockMvc;

    @Test
    void testListUsers() throws Exception {
        mockMvc.perform(get("/users"))
            .andExpect(status().isOk())
            .andExpect(view().name("users/list"));
    }
}
```

Test d’un service avec Mockito

```
@SpringBootTest
class UserServiceTest {
    @MockBean UserRepository repo;
    @Autowired UserService service;

    @Test
    void testFindAll() {
        when(repo.findAll()).thenReturn(List.of(new User("Alice"), new
User("Bob")));
        assertEquals(2, service.findAll().size());
    }
}
```

Test d’un contrôleur REST

```java
@WebMvcTest(UserRestController.class)
class UserRestControllerTest {
    @Autowired MockMvc mockMvc;
    @MockBean UserRepository repo;

    @Test
    void testGetUserById() throws Exception {
        when(repo.findById(1L)).thenReturn(Optional.of(new User("Alice")));
        mockMvc.perform(get("/api/users/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.name").value("Alice"));
    }
}
```

Tests JPA avec BDD en mémoire

```java
@DataJpaTest
class UserRepositoryTest {
    @Autowired UserRepository repo;

    @Test
    void testSaveAndFind() {
        User u = repo.save(new User("Bob", "bob@mail.com"));
        assertTrue(repo.existsByEmail("bob@mail.com"));
    }
}
```

Astuces tests :

- Pour mocker la sécurité :
  `@WithMockUser(username="admin",roles={"ADMIN"})`
- Pour injecter des données de test :
  Place un fichier `data.sql` dans `src/main/resources`.
- Pour les tests d'intégration bout-en-bout :
  Utilise `@SpringBootTest(webEnvironment = RANDOM_PORT)` et `TestRestTemplate`.

Trucs & Astuces avancées

- **Profils Spring** :
  `@Profile("dev")` sur tes beans/services pour les charger uniquement dans l'environnement correspondant.
- **Reload automatique** :
  Ajoute `spring-boot-devtools` pour le rechargement à chaud.
- **Exceptions personnalisées** :
  Crée des exceptions métiers, gérées dans un `@ControllerAdvice`.

- **Requêtes JPA avancées** :
  Utilise `@Query` pour du JPQL natif, ou des projections DTO.
- **Swagger/OpenAPI** :
  Ajoute `springdoc-openapi-ui` pour `/swagger-ui.html`.
- **WebSocket** :
  Ajoute `spring-boot-starter-websocket` :

```
@Controller
public class ChatController {
    @MessageMapping("/chat")
    @SendTo("/topic/messages")
    public Message send(Message message) { return message; }
}
```

- **Pagination REST** :
  Expose `Page<T>`, ou enveloppe le résultat dans un objet de réponse avec `totalPages`, `page`, `size`, etc.
- **Upload de fichiers** :
  Utilise `MultipartFile` côté contrôleur, stocke en BDD ou filesystem.
- **Batch** :
  `spring-boot-starter-batch` pour l'automatisation de traitements lourds.
- **Planification de tâches** :
  `@EnableScheduling` + `@Scheduled(fixedRate = 60000)` ou cron.
- **@Async** :
  Pour exécuter en asynchrone :
  `@EnableAsync` puis `@Async` sur les méthodes longues.
- **Multi-datasource** :
  Configure plusieurs `DataSource` et `@Primary` pour la principale.
- **API HATEOAS** :
  Ajoute `spring-boot-starter-hateoas` pour enrichir tes réponses REST de liens.

Ressources utiles

- Documentation officielle Spring Boot
- Spring Guides
- Baeldung Spring Boot
- Thymeleaf Documentation
- Spring Security Reference
- Spring Data JPA
- Springdoc OpenAPI
- Testcontainers pour les tests d'intégration avec BDD/Docker
- Cheat Sheets Spring