How to avoid users to use the most commonly used password in springboot ? Do you see any library or any other option to do it ?

To prevent users from using commonly used passwords in a Spring Boot application, you have two primary options depending on your security requirements and version.

1. The Modern Way: Spring Security 6.3+ (Built-in)

If you are using a recent version of Spring Boot (3.3+), this feature is now **native**. Spring Security 6.3 introduced the `CompromisedPasswordChecker`.

By default, it integrates with **HaveIBeenPwned** (a privacy-safe API that checks if a password has appeared in known data breaches).

Implementation:
Simply verify your Spring Security version and add this Bean to your configuration:

```
@Bean
public CompromisedPasswordChecker compromisedPasswordChecker() {
    // This connects to the HaveIBeenPwned API securely (only sends a hash prefix)
    return new HaveIBeenPwnedRestApiPasswordChecker();
}
```

- **Pros:** Zero maintenance; checks against 600M+ real compromised passwords.
- **Cons:** Requires outbound internet access to the API (though it is privacy-preserving).

2. The Flexible Way: Passay Library (Offline / Custom)

If you are on an older Spring Boot version, or need to run strictly **offline** without calling external APIs, **Passay** is the industry-standard library for this.

Step 1: Add Dependency

```
<dependency>
    <groupId>org.passay</groupId>
    <artifactId>passay</artifactId>
    <version>1.6.6</version>
</dependency>
```

Step 2: Implement a `DictionaryRule`
You will need a text file containing forbidden words (e.g., `common-passwords.txt`). You can download a high-quality list from the **SecLists** repository on GitHub (look for `10k-most-common.txt`).

```
import org.passay.*;
import org.passay.dictionary.WordListDictionary;
import org.passay.dictionary.ArrayWordList;
import java.io.FileReader;
import java.util.Arrays;

public void validatePassword(String password) throws Exception {

    // 1. Load the list of common passwords (files/common-passwords.txt)
    // For production, load this bean once at startup, not per request!
    FileReader fileReader = new FileReader("src/main/resources/common-passwords.txt");
    WordListDictionary commonPasswordsDict = new WordListDictionary(
        WordLists.createFromReader(new FileReader[] { fileReader })
    );

    // 2. Create the Rule
    DictionaryRule dictionaryRule = new DictionaryRule(commonPasswordsDict);

    // 3. Create the Validator
    PasswordValidator validator = new PasswordValidator(Arrays.asList(
        new LengthRule(8, 30),              // Basic length
        dictionaryRule                      // The common password check
    ));

    // 4. Validate
    RuleResult result = validator.validate(new PasswordData(password));

    if (!result.isValid()) {
        throw new RuntimeException("Password is too common: " + validator.getMessages(res
    }
}
```

3. Which one should you choose?

Feature	**Spring Security Native**	**Passay Library**
Best For	Modern apps wanting maximum security.	Apps requiring strict offline control.
Data Source	Real-time breach data (HaveIBeenPwned).	Static text file you must update manually.
Performance	Network latency (fast, but involves I/O).	Memory intensive (loads list into RAM).
Privacy	High (Sends `k-anonymity` hash, not password).	Absolute (Data never leaves server).

Recommendation

Adhere to **NIST Guidelines**:

1. **Do use** `Spring Security`'s `CompromisedPasswordChecker` if allowed. It protects against real-world breaches, not just "common" words.
2. **Do NOT** force arbitrary complexity (e.g., "Must contain 1 uppercase, 1 symbol"). Instead, block specific weak passwords using the methods above.[1] [2]

⁂

1. https://netwrix.com/en/resources/blog/nist-password-guidelines/
2. https://linfordco.com/blog/nist-password-policy-guidelines/
3. https://github.com/spring-projects/spring-security/issues/15305
4. https://dimitri.codes/spring-security-compromisedpasswordchecker/
5. https://dzone.com/articles/spring-boot-custom-password-validator-using-passay
6. https://www.javacodegeeks.com/spring-security-6-3-whats-new.html
7. https://www.passay.org/javadocs/org/passay/DictionaryRule.html
8. https://www.psmpartners.com/blog/nist-password-best-practices/
9. https://spring.io/blog/2025/03/19/spring-security-6-3-8-6-4-4-are-now-available
10. https://www.baeldung.com/java-passay
11. https://www.baeldung.com/spring-security-6-3
12. https://www.tutorialspoint.com/passay/passay_dictionaryrule.htm
13. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/429036f4-3bc3-4a7a-8c6f-90246e73b9b9/image.jpg
14. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/24f18c0a-b56e-4862-a619-e2959036a5c2/image.jpg
15. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/97e9ed86-ed11-4035-a18b-1eb05c1bab4e/image.jpg
16. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/87f62423-96f2-4071-9802-8f6699e0ecd8/image.jpg
17. https://www.baeldung.com/spring-security-detect-compromised-passwords
18. https://docs.enterprise.spring.io/spring-security/docs/6.3.12/api/org/springframework/security/authentication/password/CompromisedPasswordChecker.html
19. https://docs.spring.vmware.com/spring-security/reference/whats-new.html