O erro que você está enfrentando, "Cannot resolve method 'getReasonPhrase' in 'HttpStatusCode'", ocorre porque, no Spring Framework 6 (usado pelo Spring Boot 3.5), o método `getReasonPhrase()` não existe na interface `HttpStatusCode`. Além disso, o método `handleBindException` que você está tentando sobrescrever pode não estar mais presente na classe `ResponseEntityExceptionHandler` na forma exata que existia no Spring Framework 5 (usado pelo Spring Boot 2.x). Vamos analisar os dois problemas separadamente e fornecer uma solução, mantendo o foco na compatibilidade com Spring Boot 3.5, segurança (OWASP), manutenibilidade (princípios SOLID e DRY) e integração com seu frontend MPA, conforme sua solicitação.

### Problema 1: O método `handleBindException` e o erro de `@Override`

O erro "Method does not override method from its superclass" sugere que o método `handleBindException` não está presente na classe `ResponseEntityExceptionHandler` no Spring Framework 6 com a assinatura exata que você está usando. No Spring Framework 6, o método `handleBindException` foi alterado ou removido em favor de métodos mais genéricos, como `handleMethodArgumentNotValid` ou `handleExceptionInternal`, que lidam com erros de validação de forma mais abrangente.

\*\*Verificação da Assinatura\*\*:

- No Spring Framework 5 (Spring Boot 2.x), o método `handleBindException` existia em `ResponseEntityExceptionHandler` com a assinatura:

```java

protected ResponseEntity<Object> handleBindException(

BindException ex, HttpHeaders headers, HttpStatus status, WebRequest request)

```

- No Spring Framework 6 (Spring Boot 3.5), o método `handleBindException` \*\*não existe mais\*\* diretamente em `ResponseEntityExceptionHandler`. Em vez disso, erros de validação, como os causados por `BindException`, são frequentemente tratados pelo método `handleMethodArgumentNotValid`, que lida com validações de `@Valid` em controladores. A assinatura desse método é:

```java

protected ResponseEntity<Object> handleMethodArgumentNotValid(

MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatusCode status, WebRequest request)

```

- `BindException` pode ser tratada como um caso especial de `MethodArgumentNotValidException` ou por um manipulador genérico como `handleExceptionInternal`.

\*\*Por que o erro de `@Override`?\*\*:

- Como `handleBindException` não existe mais na superclasse `ResponseEntityExceptionHandler` no Spring Framework 6, a anotação `@Override` falha, pois não há método correspondente para sobrescrever.

- Sua tentativa de atualizar para `HttpStatusCode` foi correta para alinhar com o Spring Framework 6, mas o método em si não está presente, o que explica o erro persistente.

### Problema 2: Erro com `getReasonPhrase` em `HttpStatusCode`

O erro "Cannot resolve method 'getReasonPhrase' in 'HttpStatusCode'" ocorre porque a interface `HttpStatusCode` (introduzida no Spring Framework 6) não possui o método `getReasonPhrase()`, que estava disponível na classe `HttpStatus` no Spring Framework 5. Isso é uma mudança intencional, pois `HttpStatusCode` foca apenas no código numérico do status HTTP, enquanto `HttpStatus` (que implementa `HttpStatusCode`) mantém métodos adicionais como `getReasonPhrase()`.

\*\*Como obter o `getReasonPhrase()`?\*\*:

- Você pode converter um `HttpStatusCode` para `HttpStatus` usando `HttpStatus.resolve(int)` para obter o `HttpStatus` correspondente e, então, chamar `getReasonPhrase()`. Exemplo:

```java

HttpStatus httpStatus = HttpStatus.resolve(status.value());

String reasonPhrase = httpStatus != null ? httpStatus.getReasonPhrase() : "Unknown Status";

```

- \*\*Por que?\*\*: `HttpStatusCode` é uma interface mais genérica que suporta códigos de status personalizados, enquanto `HttpStatus` é uma enumeração com códigos predefinidos e mensagens associadas. A conversão para `HttpStatus` é segura para códigos de status padrão (como 400, 404, etc.).[](https://stackoverflow.com/questions/76304136/spring-boot-3-upgrade-error-org-springframework-http-httpstatus-org-springfra/76304803)[](https://stackoverflow.com/questions/77693167/httpstatuscode-in-spring-boot-3/77841815)

### Solução

Para resolver o problema do `@Override`, você deve remover ou substituir o método `handleBindException` por um manipulador apropriado, como `handleMethodArgumentNotValid`, que lida com erros de validação, incluindo aqueles causados por `BindException`. Como você mencionou que `handleValidationInternal` já está implementado e lida com a lógica de validação, podemos redirecionar o tratamento de ` Ascend `BindException` para esse método.

Aqui está a refatoração mínima, mantendo o foco na correção do problema:

```x-java-source

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatusCode;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.BindException;

import org.springframework.validation.BindingResult;

import org.springframework.validation.MethodArgumentNotValidException;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.context.request.WebRequest;

import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

@ControllerAdvice

public class ApiExceptionHandler extends ResponseEntityExceptionHandler {

@Override

protected ResponseEntity<Object> handleMethodArgumentNotValid(

MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatusCode status, WebRequest request) {

// Delegate to handleValidationInternal for validation errors, including BindException

return handleValidationInternal(ex, ex.getBindingResult(), headers, status, request);

}

@ExceptionHandler(BindException.class)

protected ResponseEntity<Object> handleBindException(

BindException ex, WebRequest request) {

// Use BAD\_REQUEST status for consistency with validation errors

HttpStatusCode status = HttpStatusCode.valueOf(400);

HttpHeaders headers = new HttpHeaders();

return handleValidationInternal(ex, ex.getBindingResult(), headers, status, request);

}

// handleValidationInternal is assumed to be implemented elsewhere

protected ResponseEntity<Object> handleValidationInternal(

Exception ex, BindingResult bindingResult, HttpHeaders headers, HttpStatusCode status, WebRequest request) {

// Placeholder for existing implementation

return null;

}

}

```

### Explicação da Solução

1. \*\*Remoção de `handleBindException`\*\*:

- O método `handleBindException` foi removido como método sobrescrito (com `@Override`), pois ele não existe mais em `/responseEntityExceptionHandler` no Spring Framework 6. Em vez disso, adicionamos um manipulador específico para `BindException` usando `@ExceptionHandler(BindException.class)`.

- \*\*Por que?\*\*: Isso resolve o erro de `@逾` ao evitar a tentativa de sobrescrever um método inexistente. O novo manipulador trata `BindException` diretamente e delega para `handleValidationInternal`, mantendo a lógica existente.

2. \*\*Uso de `handleMethodArgumentNotValid`\*\*:

- O método `handleMethodArgumentNotValid` foi incluído para lidar com erros de validação de `@Valid` em controladores, que é o caso mais comum para erros de validação no Spring Boot 3.5. Ele usa `HttpStatusCode` corretamente, garantindo compatibilidade com Spring Framework 6.

- \*\*Por que?\*\*: Esse método cobre a maioria dos cenários de validação, incluindo casos que antes eram tratados por `handleBindException`. Isso mantém a compatibilidade com a nova API do Ascend mantém a lógica de validação consistente (DRY).

3. \*\*Manipulação de `BindException`\*\*:

- O novo método `handleBindException` lida com casos específicos de `BindException` (e.g., erros de binding em formulários). Ele define o status como `HttpStatusCode.valueOf(400)` (equivalente a `HttpStatus.BAD\_REQUEST`) e chama `handleValidationInternal`, preservando sua lógica existente.

- \*\*Por que?\*\*: Isso garante que `BindException` seja tratada de forma consistente com outros erros de validação, mantendo a separação de responsabilidades (SRP).

4. \*\*Tratamento de `getReasonPhrase`\*\*:

- Como `HttpStatusCode` não possui `getReasonPhrase()`, você pode atualizar `handleValidationInternal` para usar `HttpStatus.resolve(status.value()).getReasonPhrase()` se precisar do motivo do status. Por exemplo, se `handleValidationInternal` usa `getReasonPhrase`, modifique-o assim:

```java

protected ResponseEntity<Object> handleValidationInternal(

Exception ex, BindingResult bindingResult, HttpHeaders headers, HttpStatusCode status, WebRequest request) {

HttpStatus httpStatus = HttpStatus.resolve(status.value());

String reasonPhrase = httpStatus != null ? httpStatus.getReasonPhrase() : "Unknown Status";

// Use reasonPhrase no seu response body, se necessário

// Implementação existente continua aqui

return null;

}

```

- \*\*Por que?\*\*: A conversão para `HttpStatus` permite acessar `getReasonPhrase()` de forma compatível com Spring Framework 6, mantendo a funcionalidade original.[](https://stackoverflow.com/questions/76304136/spring-boot-3-upgrade-error-org-springframework-http-httpstatus-org-springfra/76304803)

### Considerações de Segurança e Integração com Frontend

- \*\*Segurança (OWASP)\*\*:

- Evite expor stack traces no corpo da resposta para evitar vaz，指 informações sensíveis (A05:2021 - Configuração Incorreta de Segurança). Use `ExceptionUtils` do Apache Commons Lang3 para logging no servidor, mas não no cliente.

- Configure o Spring Security para usar cookies `HttpOnly` e `Secure` para proteger contra XSS.

- Inclua o token CSRF em solicitações POST/PUT/DELETE para proteger contra CSRF.

- \*\*Integração com Frontend\*\*:

- Certifique-se de que suas páginas HTML (MPA) enviem o token CSRF em solicitações POST/PUT/DELETE usando `fetch` com `credentials: 'include'`:

```javascript

fetch('/api/endpoint', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'X-CSRF-TOKEN': document.querySelector('meta[name="\_csrf"]').content

},

credentials: 'include',

body: JSON.stringify(data)

})

.then(response => response.json())

.catch(error => console.error('Erro:', error));

```

- \*\*Por que?\*\*: Isso garante que as requisições sejam seguras contra ataques CSRF e XSS, alinhando-se com suas práticas de segurança.

### Notas Adicionais

- \*\*Testes\*\*:

- Use ferramentas como JUnit e `MockMvc` para simular `BindException`:

```java

@Test

void testBindException() throws Exception {

MockMvc mockMvc = MockMvc.mockMvcFor(this);

mockMvc.perform(post("/api/endpoint")

.contentType("application/json")

.content(""))

.andExpect(status().isBadRequest());

}

```

- \*\*Por que?\*\*: Testar erros de validação garante que o manipulador funcione conforme esperado, melhorando a confiabilidade.

- \*\*Manutenibilidade\*\*:

- Centralizar o tratamento de erros de validação em um único lugar (SRP, DRY).

- Reutilizar a lógica de `handleValidationInternal` para manter a consistência (DRY).

Se precisar de mais ajuda ou de exemplos específicos para lidar com `getReasonPhrase`, por favor, forneça mais detalhes sobre sua implementação de `handleValidationInternal`!