FIAP - FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA

JAVA ADVANCED

- João Carlos Lima e Silva

TURMA:

- 2TDSPZ

ALUNOS:

- Felipe Torlai RM 550263
- Felipe Pinheiro RM 550244
- Gabriel Girami RM 98017
- Gustavo Vinhola RM 98826
- Jean Carlos RM 550430

EMPRESA:

- NextGen

PROJETO:

- Sistema de Gestão de Experiência do Cliente

OBJETIVO:

- Mostrar as alterações feitas da Sprint 1 para a Sprint 2

SUMÁRIO

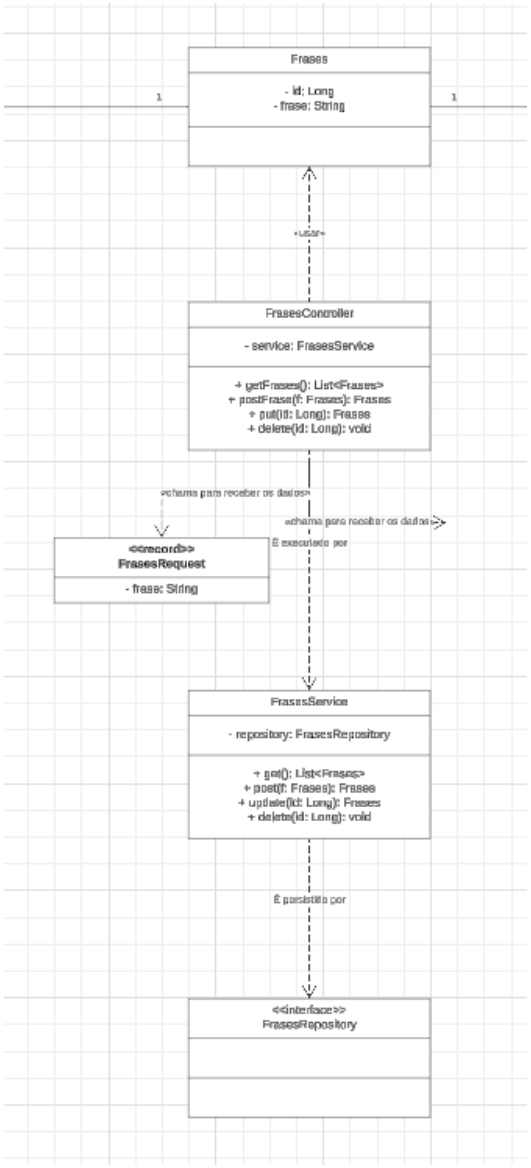
1. CLASSES NOVAS

A primeira e mais crucial mudança foi no diagrama de classes, no qual foi adicionada mais uma entidade chamada “frases”, cujo objetivo é guardar as frases que o usuário colocou em seu feedback. Essa entidade vai possuir dois atributos:

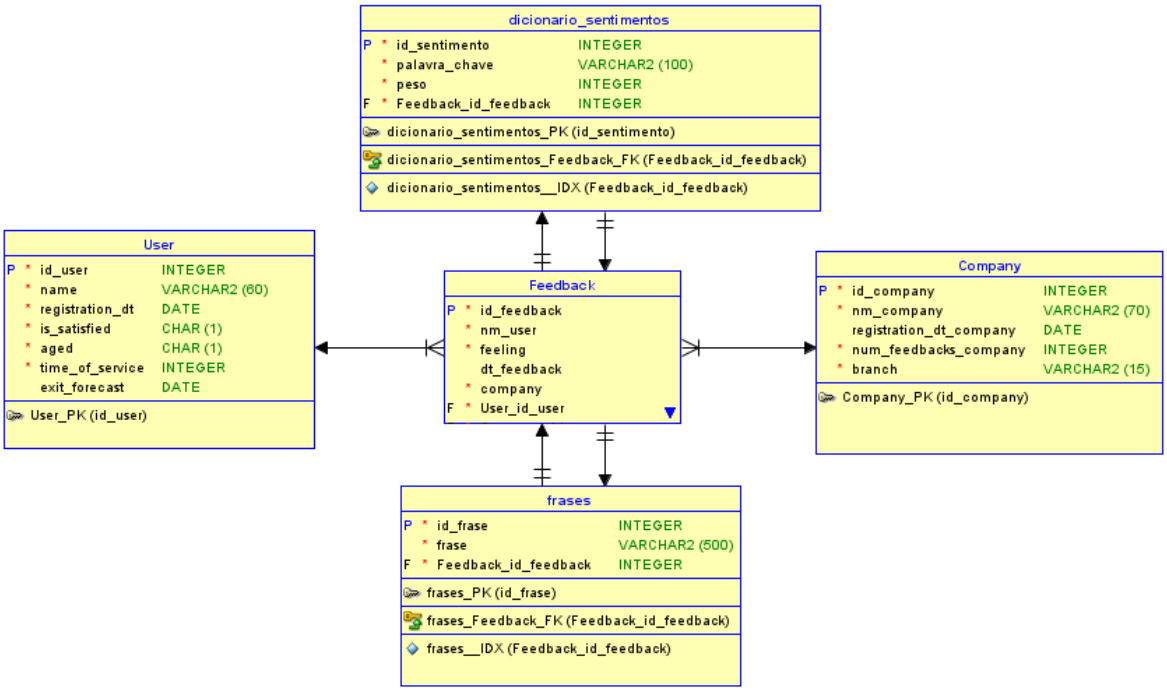
- ✓ Id: atributo Long com o objetivo de identificar a frase
- ✓ Frase: atributo String que irá guardar as frases escritas pelo usuário

Já que vamos fornecer as requisições HTTP para essa entidade, as classes de controle, repositório e serviço também foram adicionadas. Essa entidade foi adicionada de acordo com a sugestão do professor de banco de dados:

Mudança no Diagrama de Classes:



Mudança no Diagrama Entidade-Relacionamento:



Entidade adicionada:

```java
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Frases {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    private String frase;
}
```

FrasesRequest adicionado:

```
1  + package com.fiap.nextgen.DTO;
2  +
3  + import jakarta.persistence.GeneratedValue;
4  + import jakarta.persistence.GenerationType;
5  + import jakarta.persistence.Id;
6  + import jakarta.validation.constraints.NotBlank;
7  +
8  + public record FrasesRequest(
9  +
10 +     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
11 +     Long id,
12 +
13 +     @NotBlank
14 +     String frase
15 +
16 + ) {}
```

Controller adicionado:

```
26 + @RestController
27 + @Slf4j
28 + @RequestMapping("frases")
29 + public class FrasesController {
30 +
31 +     @Autowired
32 +     FrasesService frasesService;
33 +
34 +     @GetMapping()
35 +     public List<Frases> getFrases() {
36 +         log.info("Pegando todas as frases");
37 +         return frasesService.getAll();
38 +     }
39 +
40 +     @PostMapping()
41 +     @ResponseStatus(CREATED)
42 +     public Frases postFrase(@RequestBody FrasesRequest fraseRequest) {
43 +         log.info("Postando um texto");
44 +         return frasesService.create(fraseRequest);
45 +     }
46 +
47 +     @PutMapping("{id}")
48 +     public Frases put(@PathVariable Long id, @RequestBody FrasesRequest frasesRequest) {
49 +         log.info("Atualizando  as frases");
50 +         return frasesService.atualizar(id, frasesRequest);
51 +     }
52 +
53 +     @DeleteMapping("{id}")
54 +     @ResponseStatus(NO_CONTENT)
55 +     public void delete(@PathVariable Long id) {
56 +         log.info("Apagando as frases criadas");
57 +         frasesService.delete(id);
58 +     }
59 + }
```

Service adicionado:

```
+ @Service
+ public class FrasesService {
+
+     @Autowired
+     FrasesRepository frasesRepository;
+
+     public List<Frases> getAll() {
+         return frasesRepository.findAll();
+     }
+
+     public Frases create(FrasesRequest fraseRequest) {
+         Frases frase = constructFrases(fraseRequest);
+         return frasesRepository.save(frase);
+     }
+
+     public Frases atualizar(Long id, FrasesRequest fraseRequest) {
+         Frases frases = constructFrases(fraseRequest);
+         checkExistence(id);
+         frases.setId(id);
+         return frasesRepository.save(frases);
+     }
+
+     public void delete(Long id) {
+         frasesRepository.deleteById(id);
+     }
+
+
+     public Frases checkExistence(Long id) {
+         return frasesRepository
+             .findById(id)
+             .orElseThrow(() -> new ResponseStatusException(NOT_FOUND, "Não existe um usuário com este ID"));
+     }
+
+     public Frases constructFrases(FrasesRequest frasesRequest) {
+         return new Frases(
+             frasesRequest.id(),
+             frasesRequest.frase()
+         );
+     }
+
+ }
```

Repository adicionado:

```
+ package com.fiap.nextgen.Repository;
+
+ import org.springframework.data.jpa.repository.JpaRepository;
+
+ import com.fiap.nextgen.Model.Frases;
+
+ public interface FrasesRepository extends JpaRepository<Frases, Long> {
+
+ }
```

2. NOVO DICIONÁRIO (ENUM)

Também pela sugestão do professor do banco de dados, um novo enum vai ser criado, chamado “Dicionário de Sentimentos”, cujo objetivo é ser realmente um dicionário com sentimentos padrão que o usuário vai ter com a empresa, contendo 5 sentimentos, dentre eles:

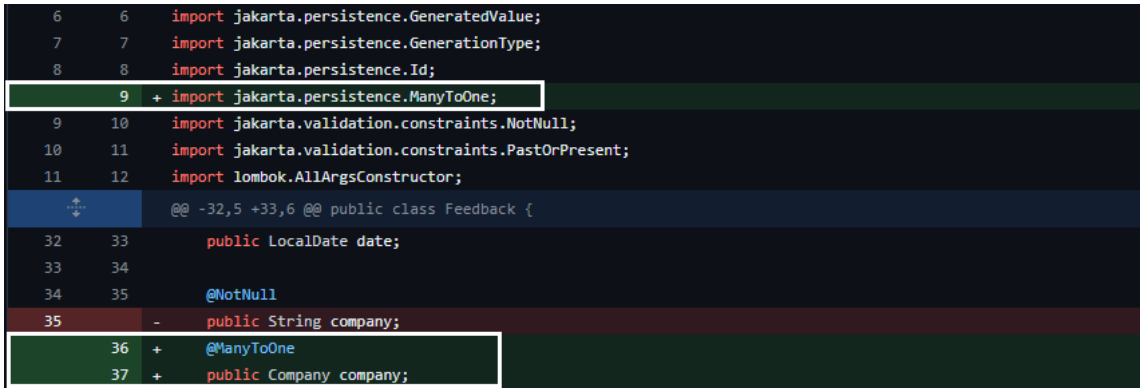
- ✓ Terrible
- ✓ Bad
- ✓ Regular
- ✓ Good
- ✓ Awesome

Foi criado um atributo, dois construtores e um método GET, veja as alterações no código:

```
+ package com.fiap.nextgen.Model;
+
+ public enum DicionarioSentimentos {
+
+     TERRIBLE(1),
+     BAD(2),
+     REGULAR(3),
+     GOOD(4),
+     AWESOME(5);
+
+     private int feeling;
+
+     DicionarioSentimentos(int feeling) {
+         this.feeling = feeling;
+     }
+
+     DicionarioSentimentos() {
+
+     }
+
+     public int getFeeling() {
+         return feeling;
+     }
+ }
```

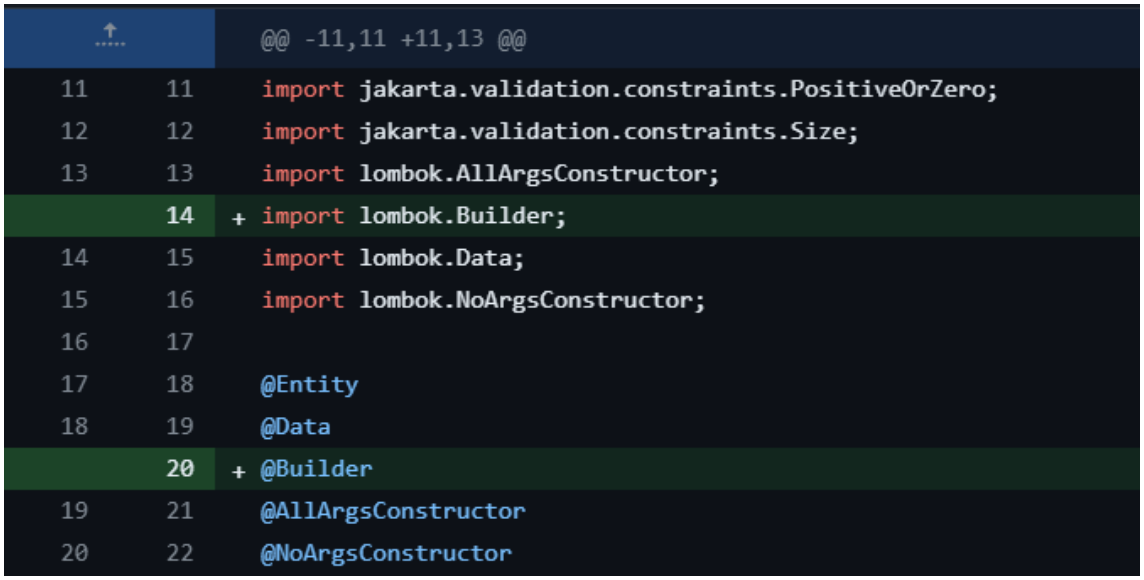
3. RELACIONAMENTO

A terceira alteração feita foi nos relacionamentos. A classe model “Feedback” tem um atributo chamado “Company” que é uma Chave Estrangeira (Foreign Key). Para a entrega da Sprint 2, adicionamos esse relacionamento com as anotações adequadas:



4. BUILDER

A quarta alteração foi a implementação do Design Pattern Builder, que nos permite a criação de dados automaticamente quando rodamos nosso aplicativo Spring. Para essa implementação, foi adicionado nas 3 classes model (Users, Feedback, Company) a notação Builder:

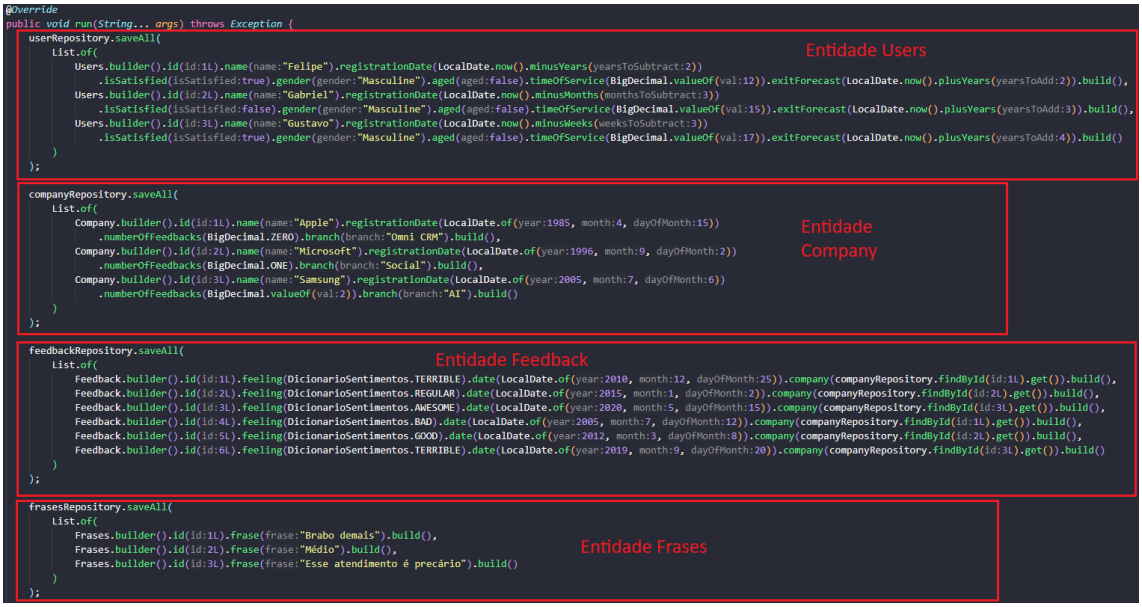


Após, foi adicionado uma classe chamada DatabaseSeeder cujo objetivo é automatizar a criação dos dados no runtime:

1. Configuração e Injeção de Dependências:

```java
@Configuration
public class DatabaseSeeder implements CommandLineRunner {


    @Autowired
    UserRepository userRepository;

    @Autowired
    CompanyRepository companyRepository;

    @Autowired
    FeedbackRepository feedbackRepository;

    @Autowired
    FrasesRepository frasesRepository;

```

2. Builders de cada entidade:



Obs: Dê um zoom se necessário

5. PAGINAÇÃO

A quinta alteração feita foi a inclusão da paginação no nosso sistema usando o Pageable e suas anotações:

Alteração no FeedbackController:

```
@GetMapping
public List<Feedback> getMethod(
    @RequestParam(required = false) String company,
    @RequestParam(required = false) Integer mes,
    @PageableDefault(size = 5, sort = "date", direction = Direction.DESC) Pageable pageable
) {
    log.info(msg:"Pegando os feedbacks...");
    return feedbackService.getAllFeedbacks(company, mes, pageable);
}
```

Alteração no FeedbackService:

```
public List<Feedback> getAllFeedbacks(
    @RequestParam(required = false) String company,
    @RequestParam(required = false) Integer mes,
    @PageableDefault(size = 5, sort = "date", direction = Direction.DESC) Pageable pageable
) {

    if (mes != null && company != null) {
        return feedRepository.findByCompanyNameAndMes(company, mes, pageable);
    }

    if (mes != null) {
        return feedRepository.findByMes(mes, pageable);
    }

    if (company != null) {
        return feedRepository.findByCompanyName(company, pageable);
    }

    return feedRepository.findAll();
}
```

6. SISTEMA DE CACHE

A sexta alteração foi do sistema de memória cache no sistema, ele foi implementado nas entidades Users e Company, as quais não tem criações com grande frequência. Para tudo funcionar, algumas classes devem ter suas anotações colocadas:

1. Classe main (NextgenApplication.java):

```
2   2
3   3   import org.springframework.boot.SpringApplication;
4   4   import org.springframework.boot.autoconfigure.SpringBootApplication;
    5 + import org.springframework.cache.annotation.EnableCaching;
5   6
6   7   @SpringBootApplication
    8 + @EnableCaching
7   9   public class NextgenApplication {
8   10
9   11          public static void main(String[] args) {
10  12                  SpringApplication.run(NextgenApplication.class, args);
11  13          }
12  14
13  15  }
```

2. Classe CompanyController

```
@RestController
@Slf4j
@RequestMapping(path = "companies")
@CacheConfig(cacheNames = "companies")
public class CompanyController {
```

```
@PostMapping
@ResponseStatus(CREATED)
@CacheEvict(allEntries = true)
public Company postMethod(@RequestBody @Valid CompanyRequest companyRequest) {
    log.info(msg:"Cadastrando uma empresa...");
    return companyService.createCompany(companyRequest);
}

@PutMapping("{id}")
@CacheEvict(allEntries = true)
public Company putMethod(@PathVariable Long id, @RequestBody @Valid CompanyRequest companyRequest) {
    log.info("Atualizando a empresa com o id " + id);
    return companyService.updateCompany(id, companyRequest);
}

@DeleteMapping("{id}")
@ResponseStatus(NO_CONTENT)
@CacheEvict(allEntries = true)
public void deleteMethod(@PathVariable @Valid Long id) {
    log.info(msg:"Deletando a empresa");
    companyService.deleteCompany(id);
}
```

7. SPRINGDOC

A próxima alteração foi a implementação da documentação automática Springdoc com Swagger UI. Para isso, foi feita a instalação da dependência no arquivo pom.xml:

```
+		<dependency>
+			<groupId>org.springdoc</groupId>
+			<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
+			<version>2.5.0</version>
+		</dependency>
+
```

Depois disso, foi colocado o título da aplicação, descrição, versão e contato no arquivo principal (NextgenApplication.java):

```
+ @OpenAPIDefinition(
+		info = @Info(
+			title = "NextGen",
+			summary = "API do App NextGen",
+			description = "Sistema de Gestão de Experiência do Cliente",
+			version = "1.0.0",
+			contact = @Contact(
+				name = "Felipe Pinheiro",
+				email = "fsp12371@gmail.com"
+			)
+		)
+ )
```

Logo após, foi colocado as descrições e corpos de resposta para cada método em todos os controllers:

- Método GET:

```
+    @Operation(
+        summary = "Listar Feedbacks",
+        description = "Retorna um array com todos os atributos do feedback"
+    )
+    @ApiResponses({
+        @ApiResponse(responseCode = "200", description = "Feedback retornado com sucesso!"),
+        @ApiResponse(responseCode = "401", description = "Feedback não autorizado. Realize a autenticação em /login")
+    })
```

- Método POST:

```
+    @Operation(
+        summary = "Cadastrar feedback",
+        description = "Cadastro de um feedback com o corpo de uma requisição"
+    )
+    @ApiResponses({
+        @ApiResponse(responseCode = "200", description = "Feedback cadastrado com sucesso!"),
+        @ApiResponse(responseCode = "400", description = "Validação falhou. Verifique os dados enviados no corpo da requisição"),
+        @ApiResponse(responseCode = "401", description = "Não autorizado. Realize a autenticação em /login")
+    })
```

- Método PUT:

```
+    @Operation(
+        summary = "Atualizar empresa",
+        description = "Atualiza os dados da empresa com o id informado na path"
+    )
+    @ApiResponses({
+        @ApiResponse(responseCode = "200", description = "Empresa atualizada com sucesso!"),
+        @ApiResponse(responseCode = "400", description = "Validação falhou. Verifique os dados enviados no corpo da requisição"),
+        @ApiResponse(responseCode = "401", description = "Não autorizado. Realize a autenticação em /login"),
+        @ApiResponse(responseCode = "404", description = "Não existe empresa com o `id` informado")
+    })
```

- Método DELETE:

```
+    @Operation(
+        summary = "Apagar bloco de frases",
+        description = "Apaga a frase com o id informado no parâmetro de path"
+    )
+    @ApiResponses({
+        @ApiResponse(responseCode = "204", description = "Frase apagada com sucesso!"),
+        @ApiResponse(responseCode = "401", description = "Não autorizado. Realize a autenticação em /login")
+    })
```

- Método Get by ID:

```
+    @Operation(
+        summary = "Pegar empresa pelo id",
+        description = "Retorna os dados da empresa com o id informado no parâmetro de path"
+    )
+    @ApiResponses({
+        @ApiResponse(responseCode = "200", description = "Empresa retornada com sucesso!"),
+        @ApiResponse(responseCode = "401", description = "Não autorizado. Realize a autenticação em /login"),
+        @ApiResponse(responseCode = "404", description = "Não existe empresa com o `id` informado")
+    })
```