Construção de Sistemas de Software 2022/23

Relatório Projeto

- Fase 1-

-Tomás Piteira, 56303
-Daniel Lopes, 56357
-Miguel Ramos, 56377

Índice

Modelo de Domínio

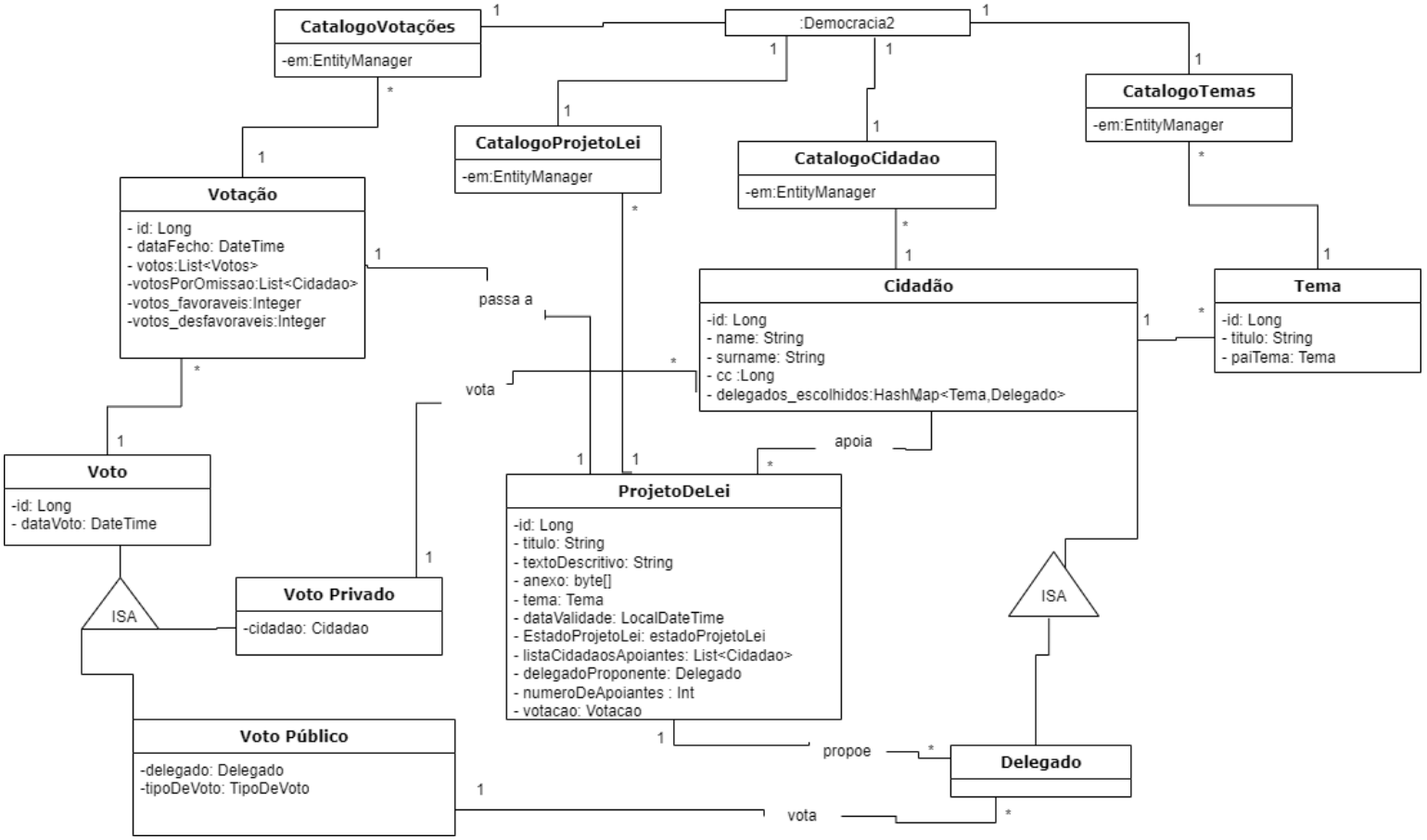
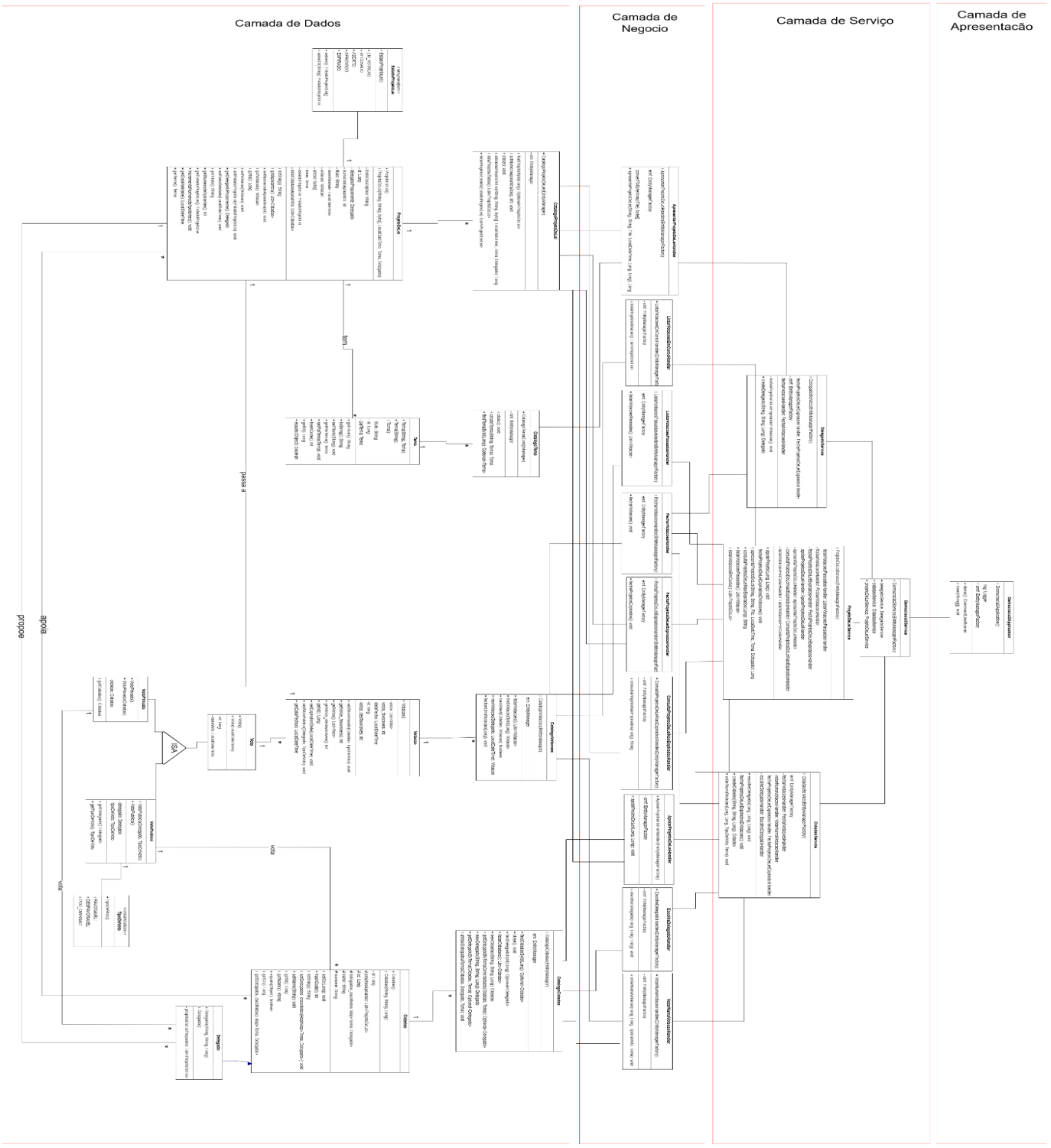
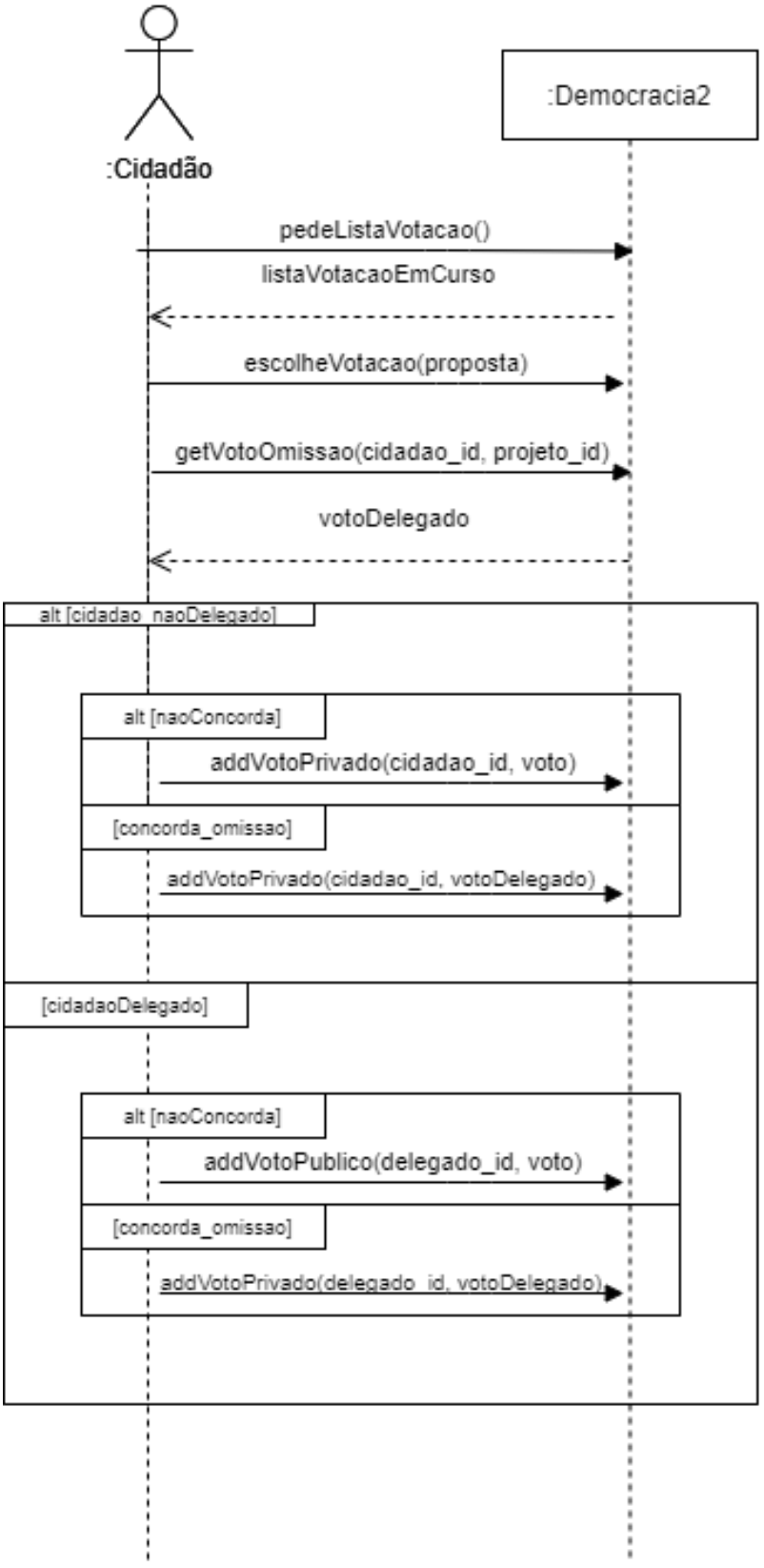


Diagrama de Classes



NOTA: Diagrama colocado na pasta Image

Caso de Uso J: Votar numa Proposta



Decisões de Implementação

Estrutura:

O nosso grupo decidiu dividir a aplicação Democracia2 em 4 camadas: Camada de Apresentação, Camada de Serviço/Aplicação, Camada de Negócio e Camada de Dados.
Como nesta aplicação o processamento aos casos de uso envolvem múltiplos recursos transacionais preferimos criar a Camada de Aplicação. Esta tem duas secções a Democracia2Service que é a Camada de Aplicação mais "acima" enquanto que as Camadas CidadaoService, DelegadoService e ProjetoDeLeiService estão mais "abaixo".
Foram usados Catálogos e Handlers devido a nossa utilização anterior em DCO, o que nos permitiu um maior à vontade e experiência. Foram usados EntityManager ao invés de repositórios pois conseguimos ter uma maior flexibilidade e controle sobre as operações de persistência. Embora da desvantagem de termos que escrever o código o que no caso dos repositórios não aconteceria, conseguimos realizar operações complexas diretamente.

Cidadão

A classe Cidadao é uma entidade JPA, que representa um cidadão no sistema Democracia2.

Anotações usadas:

-@Entity : indica que a classe é uma entidade JPA

-@Table : especificar o nome da tabela

-@Inheritance : usada a estrutura de herança, será explicada mais abaixo no relatório

-@Id e @GeneratedValue para indicação que o id gerado pelo o sistema é chave primária da entidade

-@NonNull de os valores não poderem ser null

Foram usadas a seguintes anotações do JPA para realizar mapeamento o mapa dos delegados escolhidos por cada cidadão em que cada delegado é associado a um Tema deste modo:

A anotação @ManyToMany é usada para especificar a relação entre as entidades Cidadao e Delegado. A anotação @JoinTable é usada para especificar a tabela intermediária que é criada para mapear essa relação. A anotação @MapKeyJoinColumn é usada para especificar que a chave do mapa delegados_escolhidos deve ser mapeada como uma chave estrangeira para a entidade Tema.
Na classe do Cidadão é feito o mapeamento para a lista de apoiantes presente na entidade ProjetoDeLei.

```
@ManyToMany
@JoinTable(
        name = "delegados_escolhidos",
        joinColumns = @JoinColumn(name = "cidadao_id"),
        inverseJoinColumns = @JoinColumn(name = "delegado_id")
)
@MapKeyJoinColumn(name = "tema_id")
protected Map<Tema, Delegado> delegados_escolhidos;
```

Herança: Cidadão e Delegado

```
@Inheritance(strategy = InheritanceType.JOINED)
```

Inicialmente, foi pensado em criar um enumerado para diferenciar se um Cidadão era um Delegado ou não. No entanto, decidimos optar por uma estrutura de herança. Embora muitos dos atributos do Delegado sejam herdados da classe Cidadão, acreditamos que a diferenciação a nível de dados e na lógica de negócio seria melhor alcançada com a herança. Além disso, essa estrutura permitiria a adição de atributos exclusivos do Delegado no futuro.

Optamos por utilizar a estratégia JOINED, pois ela cria uma tabela para cada classe da hierarquia, incluindo uma tabela para a classe raiz (superclasse) que contém apenas as colunas compartilhadas por todas as subclasses, o que nos permite realizar uma melhor diferenciação entre Cidadão e Delegado . Além disso, como só existe uma subclasse da classe Cidadão, não há preocupação quanto à criação de muitas tabelas. Esta estratégia também garante a integridade referencial entre as tabelas, pois as chaves estrangeiras são definidas explicitamente.

Voto

```
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "type")
```

Escolhemos implementar os Votos numa single table, dado que, a facilidade de implementar a mesma, a simplicidade das classes VotoPublico e VotoPrivado , e a similaridade de ambas, pareceram-nos razões suficientes para a escolha da mesma.

O uso de @DiscriminatorColimn cria uma coluna para diferenciar as classes VotoPublico, das classes VotoPrivado, coluna essa que ficou com o nome “type”, e usamos o @DiscriminatorValue para escolher qual seria a string de diferenciação de cada uma das classes.

Herança: Voto para Voto Público e Voto Privado

Decidimos separar os Voto em dois tipos, de modo a conseguir fazer uma diferenciação de voto com conteúdo e de voto sem conteúdo.

```
@OneToOne private Delegado delegado;

@Enumerated(EnumType.STRING)
private TipoDeVoto tipoDeVoto;
```

Para isso criamos o VotoPublico, que tem um delegado associado ao mesmo, e o seu tipo de voto. Esse delegado associado fizemos com uma relação @OneToOne porque o voto só está associado a um delegado e um delegado só está associado a um voto. Por fim, usamos um @Enumerated para o Tipo DeVoto já que o dividimos implementer como um enumerado.

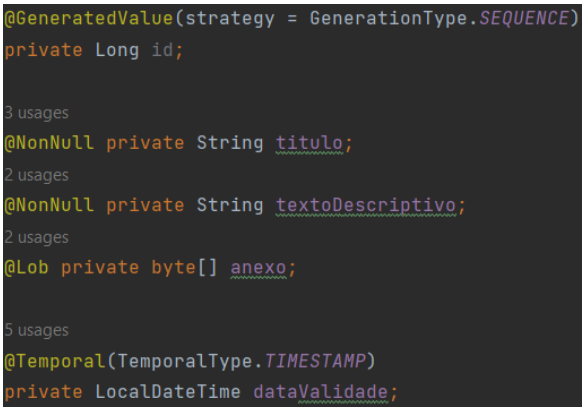
Votação

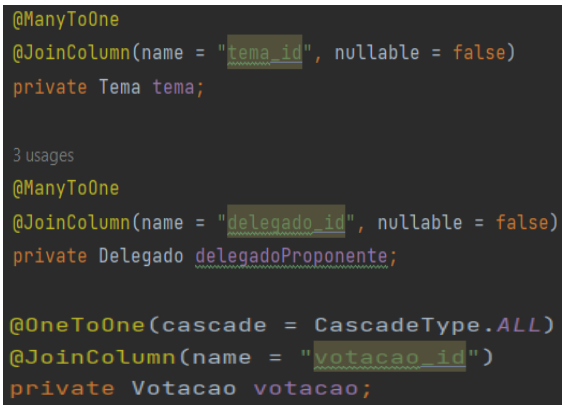
```
@OneToMany(cascade = CascadeType.ALL)
private List<Voto> votos = new LinkedList<>();
```

Implementamos uma classe Votação, para guardar as informações da votação. Decidimos que esta estaria associada a uma lista de votos e utilizamos uma relação @OneToMany uma vez que uma votação vai ter vários votos, mas cada voto só pertence a uma votação.

Projeto de Lei

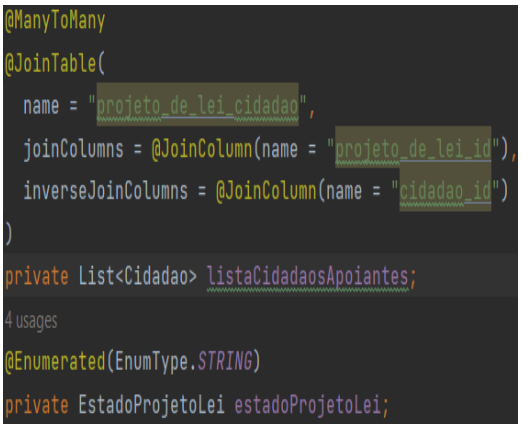
A classe ProjetoDeLei representa as propostas de lei que são criadas e que serão, posteriormente, apoiadas por cidadãos. Esta classe é anotada com **@Entity**, o que significa que irá ser mapeada para uma tabela na base de dados. Possui vários getters e setters para as as variáveis de instância e recebe dois construtores: um exigido pelo JPA e um que recebe parâmetros para o título, anexo, data de expiração, tema e delegado que propõe a proposta.





Na imagem acima à esquerda é visível a utilização do **@GeneratedValue**. Esta anotação JPA deixou-nos especificar o modo como o valor da chave primária será gerado. Optámos então pela geração da mesma usando a estratégia designada de Sequência. O uso de **@Lob** permite que o JPA fique avisado de que estamos perante um objeto grande, enquanto que o **@Temporal** permite que escolhamos o tipo de dado do atributo de um atributo data/hora. Como se vê, no nosso ponto de vista, o acertado para este projeto será o TIMESTAMP.

Na imagem acima à direita vemos a utilização de várias anotações JPA de relacionamento entre entidades. **@ManyToOne** foi escolhido para representar que a entidade Projeto de Lei possui um relacionamento muitos-para-um com as entidades Tema e Delegado. E, também, a anotação **@OneToOne** que representará um relacionamento de um-para-um entre o Projeto e a entidade Votação, com o acrescento de **CascadeType.ALL** que garante que todas as operações (persistir, atualizar, remover) serão aplicadas em cascata.



Usamos o **@Enumerated** uma vez que, na nossa implementação, o nosso Projeto de Lei poderá ter diversos estados: Aberto, Em votação, Aprovado, Recusado. **@JoinTable** é utilizada para indicar a tabela que será criada para armazenar os registros da relação muitos-para-muitos entre as entidades. **@JoinColumn** é utilizada para indicar a coluna que será utilizada como chave estrangeira na tabela criada pela anotação **@JoinTable**.

Testes

Os testes foram implementados tendo em consideração todos os casos de uso fornecidos. Tentámos ao máximo testar todos os possíveis focos de erro no nosso código, tendo estes mesmos testes sido um objeto essencial no nosso projeto, na medida em que nos ajudou a alcançar alguns problemas que, até então, não tinham sido detectados por nenhum membro do grupo.