MM - Escopo

Escopo

Esta seção aborda capacidade e limitações da atual versão da arquitetura e utiliza-se da aplicação modelo (Shield) como exemplo.

Capacidades

- **Injeção de elementos HTML através de diretivas customizadas:** a diretiva ***tl-key*** fornecida pela arquitetura tem como responsabilidade realizar as injeções de conteúdo e elementos, mutando-se conforme as tags em que é declarada. Esta diretiva esta intrinsecamente ligada as diretivas: ***tl-ref***, ***tl-object*** e ***tl-context***. Todas, em colaboração fornecem aos usuários da arquitetura: internacionalização e injeção de elementos HTML. Abaixo temos um exemplo ilustrativo de seu funcionamento.

Código HTML

```
<div class="inline-field-container">
        <input type="text" class="field-short" tl-key="name" ng-model="model.name" />
</div>
```

Código HTML gerado para o contexto de detalhe

```
<div class="inline-field-container ng-scope">
        <label>Name</label>
        <input type="text" class="field-medium ng-pristine ng-valid" tl-key="users.name" ng-model="model.0x1.
name" id="model.window.users.name.1" />
        <div class="description">User display name.</div>
        <div class="description" name="validation.name">May not be null.</div>
</div>
```

Código HTML gerado para o contexto de lista

```
<div class="inline-field-container ng-scope">
        <label>Name</label>
        <input type="text" class="field-medium ng-pristine ng-valid" tl-key="users.name" ng-model="model.0x1.
name" id="model.window.users.name.1" />
</div>
```

- **Internacionalização:** todo e qualquer elemento HTML injetado pela diretiva ***tl-key*** que contenha texto é automaticamente internacionalizado pela arquitetura utilizando os arquivos de mensagens (arquivos com extensão .properties)
- **Métodos genéricos para operações CRUD simples:** todas as entidades através da arquitetura já possuem endpoints RESTful com métodos de operações CRUD disponibilizados pela arquitetura
- **Data-binding na camada de View:** através da diretiva ***ng-model***, todo e qualquer elemento do tipo ***input*** que dispara eventos, atualiza objetos no controller Javascript automaticamente
- **Injeção de dependências:** aplicações podem disponibilizar serviços através da injeção de dependências pelo construtor da classe, tendo como único requisito que esta classe implemente uma interface que declare seus métodos públicos. Logo abaixo temos um exemplo desta implementação através de injeção de dependência utilizando construtores.

```
public interface IInjectable {
        void foo();
}

public class Injectable extends IInjectable {
        public void foo() {
                System.out.println("foo");
        }
}

public class Injected {
        public Injected(IInjectable injectable) {
                injectable.foo();
        }
}
```

- **Mapeamento objeto relacional:** através de frameworks como Hibernate, TopLink ou AO é possível mapear objetos no Java que reflitam as tabelas da base de dados.
- **Controle Transacional (limitado):** cada chamada HTTP é processada por um filtro de servlet que inicia uma transação, encaminha as operações e finaliza a transação se tudo ocorreu com sucesso. Se erros ocorrerem durante este processo toda a chamada é invalidada. Esta funcionalidade não possui suporte para transações aninhadas.
- **Padronização de elementos UI através de classes CSS:** o pacote AUI da Atlassian provê diversas classes para padronizar o laiaute de telas. Classes da arquitetura foram desenvolvidas para trabalhar em conjunto com este pacote e prover uma UI padronizada.
- **Widgets (calendário, modal panel, tab panel):** Diretivas ***tl-dialog***, ***tl-tabset***, ***tl-tab***, ***tl-calendar*** fornecem widgets completos para o desenvolvimento de telas mais complexas e podem ser facilmente incorporadas a telas já existentes.
- **Serialização/Deserialização genérica (podendo ser customizada pelo desenvolvedor):** Todas as chamadas de operações CRUD são tratadas inicialmente pela arquitetura e esta é responsável pela serialização/deserialização das chamadas. Se houver uma sobrecarga destes métodos genéricos, os argumentos recebidos por estes métodos já são serializados e deserializados após o retorno, isto facilita a vida do desenvolvedor ocultando detalhes de serialização/deserialização.
- **Validação de POJOS de forma genérica:** toda chamada para métodos genéricos que provém operações CRUD são validadas automaticamente pela arquitetura. Esta validação pode ser visualizada no snippet do código da arquitetura abaixo.

```
/**
     * Given an object of type T, apply default object validation backed by JSR311 API. If T is a valid object,
then return it,
     * otherwise throws {@link org.treelayer.commons.business.exception.BusinessException} containing a map of
constraint violations
     * as follows:
     *
     * <p>
     * {@code
     * { "exampleEntity.name" -> "javax.validation.NotNull.message",  "exampleEntity.id" -> "javax.validation.
NotNull.message" }
     * }
     * </p>
     *
     * @param model object of type T that will be validated
     *
     * @return validated object
     *
     * @throws org.treelayer.commons.business.exception.BusinessException if object is not valid
     */
    private T applyDefaultValidation(T model) throws BusinessException {
        Map<String, String> constraintViolations = validatorUtil.applyValidation(model);
        if (constraintViolations.size() > 0) {
            throw new BusinessException(HttpURLConnection.HTTP_BAD_REQUEST, constraintViolations);
        } else {
            return model;
        }
    }
```

- **Tratamento de erros de forma transparente:** exceções de negócio são lançadas utilizando a exceção customizada pela arquitetura BusinessException. Esta exceção é tratada pela camada mais externa e transforma as mensagens de erro em mensagens na tela do usuário, já internacionalizadas.

- **Persistência transparente, idependente do framework de persistência utilizado (Hibernate, TopLink, AO):** Todos os managers da arquitetura utilizam uma sólida base de classes que encapsula o framework de persistência utilizado. Com estas classes é possível abstrair as complexidades individuais de frameworks e oferecer assinaturas padrões para o desenvolvedor, não se importando com o framework utilizando.

Limitações

- **Não suporta pilha de modal:** temos na tela de Aplicações um relacionamento ManyToOne com Companhias. A tela de detalhe de aplicação constitui uma modal contendo todos os campos de aplicação e um combobox mostrando as companhias disponíveis. Com a atual arquitetura não é possível editar estas companhias a partir de uma aplicação. Para realizar esta operação seria necessário sobrepor a modal corrente de detalhe com uma modal de datalhe de companhia, controlando o fluxo de operações. A solução é abrir uma nova aba do navegador, realizar a operação e então continuar trabalhando com a aplicação em edição.

- **Não é possível redirecionar o usuário para uma tela de detalhe de outro objeto:** novamente tomando como exemplo a tela de Aplicações e Companhias, ao tentar editar uma companhia vinculada a aplicação corrent, devido a limitação de pilhas de modal, o usuário poderia ser redirecionado para a tela de detalhe da companhia. Esta operação não é suportada pois não é possível alterar as telas e abrir a modal de edição de companhia. A solução é abrir uma nova aba do navegador, realizar a operação e então continuar trabalhando com a aplicação em edição.

- **Ainda não há implementação para lançar mensagens genéricas ou de negócio na tela de lista:** nas telas de detalhe, no topo da modal, logo após a divisão de textos informativo é apresentado uma lista de erros de negócio (se houver erros, caso contrário nada é apresentado). Estes erros são comumente lançados como exceções do tipo BusinessException e tratados pela arquitetura. Esta implementação esta disponível apenas para telas de detalhe, a tela de listas ainda não possui tal funcionalidade.

- **Série de valores para elementos do tipo combobox, multipicker etc, são informadas no arquivo de internacionalização, desta forma perdendo a possibilidade de oferecer configurações para o usuário:** conforme o exemplo de código abaixo, séries são informadas diretamente no arquivo de propriedades que contem as mensagens de internacionalização. O formato {value | label} deve ser seguido para representar propriedades que envolvem séries. Como os valores são informados neste arquivo, não é possível implementar séries configuráveis pelo usuário final.

```
model.period.rule.values.1 = r|Revoked
model.period.rule.values.2 = g|Granted
```

- **Internacionalização é baseada no idioma do navegador e não em um perfil de usuário configurado:** a arquitetura baseia-se no atributo Accept-Language do protocolo HTTP 1.1 para recuperar as configurações de idioma. Conforme podemos ver na imagem abaixo, os idiomas configurados no navegador são passados neste atributo, em uma formatação que envolve separar os idiomas por vírgula e outros valores padrões (facilitando a interpretação por um regex). Se o usuário deseja mudar o idioma, muda-se o idioma no navegador e atualiza-se a página.