**Planejamento da Avaliação da Abordagem Multiplataforma**

Essa avaliação visa analisar cada contribuição do projeto de forma separada. A avaliação será dirigida por tarefas, onde cada tarefa irá explorar uma ou mais contribuições elencadas para esse projeto. Também existem as contribuições comuns a todas as tarefas, que são a modelagem dos conceitos do domínio em alto nível de abstração e em um único ambiente. As tarefas serão modeladas seguindo a arquitetura MVC (Model View Control) para manter a organização do experimento e utilizar a estratégia empregada no sistema de comércio eletrônico para o domínio de restaurante, sistema esse utilizado para avaliar a abordagem multiplataforma. As camadas da arquitetura MVC estão indicadas através de comentários deixados na modelagem genérica do projeto exemplo. A abordagem atualmente suporta a modelagem das camadas Model e Controller a camada View ainda não é suportada pela abordagem.

Para facilitar a realização das tarefas, o projeto conta com uma modelagem previamente especificada através da GPL (General-purpose language) disponível na abordagem, com exemplos montados para todos os modelos disponíveis na abordagem (deploy, plataforma, variáveis, funções globais e modelagem genérica). As plataformas declaradas e disponíveis na abordagem também seguem a arquitetura MVC, de modo que, as plataformas Android, Web e iOS tratam a camada Model e AndroidController, WebController e iOSController a camada Controller. A abordagem possui, além dos exemplos de modelagem, dicas para a realização de cada tarefa na GPL identificada com a palavra “Task” mais o seu número, exemplo Task 1. A abordagem conta com três arquivos disponíveis, sendo que o primeiro (Datatype.crossplat) tem-se as declarações dos tipos de variáveis usados pelo sistema. No segundo (Global.crossplat) tem-se as declarações das funções globais (global) e global com detalhes técnicos (globalDetails) e no arquivo principal (GPL.crossplat) a modelagem do sistema separada através do modelo Deploy, modelagem genérica e modelos de plataformas.

De forma resumida, o modelo Deploy é responsável em gerenciar em qual plataforma cada classe genérica irá rodar. A modelagem genérica irá comportar todos os conceitos do sistema, organizado através da arquitetura MVC. E, por último, os modelos de plataformas possui códigos nativos específicos de cada plataformas e podem ser adaptadas em tempo de geração de código para qualquer classe do sistema. A utilização do framework Velocity permitiu a adaptação dinâmica em tempo de geração de código, tornando a abordagem e a codificação do modelo de plataforma mais genérica. Para mais informações e modo de uso da abordagem multiplataforma ver documentação em anexo.

A abordagem multiplataforma permite a criação de padrões autorais para automação de rotinas através das funções globais, principalmente as funções GlobaisDetails. Esses padrões podem ser especificados pelo próprio desenvolvedor para generalizar rotinas para diferentes plataformas, como o caso da função globalDetails “ControllerDetailsGrid. Essa função é responsável em tratar os dados obtidos da camada Model e que serão enviados para a camada View de forma genérica, isto é, respeitando as especificidades de cada plataforma no processo de geração de código. Os parâmetros genéticos contidos nessa função ajudam a configuração para diferentes situações. Como exemplo essa classe poderá tratar desde os dados do carrinho de compras de um usuário na camada controle até os produtos apresentados para os usuários em uma categoria específica da loja. Ou também qualquer situação que seja necessário enviar dados para a camada View.

**Tarefa 1.**

**Foco**: Essa tarefa visa avaliar a seguinte contribuição do projeto: aproveitamento dos conceitos similares entre as plataformas.

**Tarefa**: Desenvolver uma classe POJO (Plain Old Java Objects) e uma classe DAO (Data Access Object ) para a função carrinho de compras de um sistema de comércio eletrônico para o domínio de restaurantes através da GPL (General Programing Language).   
A classe POJO pode ser “Order” e deve conter as propriedades do objeto, expostas no final dessa tarefa (a abordagem cria automaticamente os métodos getters e setters a partir das propriedades). A classe DAO (OrderDAO) deverá usar os métodos globais para a criação de um CRUD completo de forma automática e um método (**operation**) para tratar a inserção dos itens no carrinho de compras. O método deverá verificar se o produto já está inserido no carrinho para executar a ação de novo cadastro ou atualização. Esse método poderá ter a seguinte assinatura:  
**operation** gerenciaPedidos(idProd: int, quantity: int, sectionClient: int): string   
Após o desenvolvimento de ambas as classes (Order e OrderDAO) através da GPL, o modelo deploy deverá ser configurado para a geração de código para as três linguagens: Java, C# e Swift.  
Propriedades da classe “Pedidos”:  
 id: int

Product\_idProduct: int

ValueUnit: float

Quantity: int

ValueTotal: float

SectionClient: string

OrderFinalized: string

DateTimeOrder: dateTime

StatusFinalized: bool

OrderFinalized\_idOrderFinalized: int

ProductDelivered: bool

Obs1: Essa tarefa cria funções para a camada Model do sistema.

Obs2: O projeto contém dicas para essa tarefa identificados com a palavra Task 1

**Tarefa 2**

**Foco**: Essa tarefa visa avaliar as seguintes contribuições do projeto:  
A. Alteração das plataformas utilizadas no sistema com baixo custo; e  
B. Distribuição de funcionalidades entre as plataformas.

Obs: Essa tarefa trata a camada Controller do sistema.

**Tarefa**:   
A. O gerenciamento dos pedidos recebidos pelo restaurante é feito através da plataforma Web, os pedidos são expostos em uma tela que é automaticamente atualizado. A plataforma Web requisita um servidor conectado com a Internet que normalmente gera custos para sua manutenção. Dado a classe “OrderFinalizedAdm” disponível na abordagem e responsável por realizar o gerenciamento dos pedidos na camada Controller da arquitetura MVC, configure no modelo Deploy para que o gerenciamento dos pedidos seja executado nas plataformas móveis e não mais na plataforma Web, verifique se o código gerado está correto.   
Obs: Dicas para essa tarefa identificados com a palavra Task 2.A

B. A camada de controle que contém a função de carrinho de compras é responsável em obter o(s) item(ns) selecionado(s) pelo usuário, atualizar a lista de itens para aquele usuário e enviar os dados para a camada View (UI), permitindo o gerenciamento dos itens de seu carrinho pelo usuário. A lista de produtos do carrinho de compras pode ser armazenada de várias formas, entre elas: mecanismos de persistência em banco de dados ou memória (tabela Hash). Essas duas formas de armazenamento estão disponíveis nas classes: “CartControllerServerPersistence” e “CartControllerHashMapPersistence” modeladas de forma genérica na abordagem para todas as plataformas declaradas. No modelo deploy, gerencie o modo como cada plataforma irá tratar o armazenamento dos itens do carrinho de compras e verifique se o código gerado está correto;

Obs: Dicas para essa tarefa identificados com a palavra Task 2.B

C. Crie uma classe para explorar as funções globais (ControllerGPS, getLocation) responsáveis em obter os dados de localização física do usuário (GPS) implementadas nos modelos de plataformas AndroidController e iOSController. A função global ControllerGPS é responsável em implementar detalhes técnicos específicos de plataformas para obtenção dos dados de GPS. A função global getLocation é responsável em popular uma lista de strings (dataRealLocationList) com os dados de localização definidas na classe GPSPositionGetDataLocation. A classe deverá ter um método (operation) para tratar os dados de localização (sugestão de nome: useData) armazenados na lista GPSPositionGetDataLocation. Para efeito de tratamento, construa um laço e salve cada informação de localização em uma variável. No final direcione a função GPS para as plataformas móveis no modelo deploy e verifique o código gerado. Para mais detalhes veja C.1.2- GPSPosition na documentação.

Obs: Dicas para essa tarefa identificados com a palavra Task 2.C

**Tarefa 3**

**Foco**: Essa tarefa visa avaliar a seguinte contribuição do projeto: independência de evolução tecnológica. Capacidade do desenvolvedor estender a abordagem evitando a dependência das liberações do fabricante.  
  
**Tarefa**: Existe disponível uma função global implementada em cada modelo de plataforma para a camada Model (Android, Web e iOS) com a especificação da conexão de um determinado banco de dados (Web: Mysql, iOS e Android: SqLite). Crie uma função global acrescentando a rotina de conexão de um outro banco de dados, assim como MySQL, PostgreSQL, Oracle, MariaDB ou SQL Server para a plataforma Web e/ou Realm para as plataformas Android e iOS. Uma vez implementado a rotina de conexão crie uma classe através da GPL e faça um método com alguma rotina de interação com o novo banco. Como sugestão pode ser a autenticação de um usuário através de login e senha. Pode escolher a inclusão de apenas um banco de dados de preferência, como exemplo SQL Server para Web.

Obs: Dicas para essa tarefa identificados com a palavra Task 3.

**Tarefa 4**

**Foco**: Essa tarefa visa avaliar a seguinte contribuição do projeto: reuso das funções globais e conceitos em outro domínio.  
  
**Tarefa**: Tentar aplicar os conceitos modelados através da GPL, assim como as funções globais em um sistema de domínio distinto. Como sugestão, pode implementar uma função de autenticação para um sistema de biblioteca e/ou uma função para registro de um empréstimo de livro utilizando as funções globais e classes genéricas modeladas.   
**Dicas:** Para o registro de empréstimo é necessário criar duas classe, uma com as funções de um POJO (Emprestimo) e outra com as funções DAO (EmprestimoDAO). Na primeira classe poderia criar os seguintes atributos id int, idLivro int, idUser int, data de empréstimo dateTime, devolvido bool e observação string. Na classe DAO chamar as funções globais de insertObject, UpdateObject e SelectObject e criar uma função para armazenar os empréstimos (recordLoan) e outra para atualizar em caso de devolução do livro (updateLoan).

Obs: Dicas para essa tarefa identificados com a palavra Task 4.

**Tarefa 5**

**Foco**: Essa tarefa visa avaliar a seguinte contribuição do projeto: inclusão de novas plataformas no sistema com menor impacto no custo.

**Tarefa**: Criar uma função para um aplicativo smartwatch visando exibir os produtos disponíveis no smartphone. A comunicação dos dados entre os dispositivos será feito via bluetooth através da API nativa *DataClient*, o algoritmo dessa função será fornecido para agilizar a implementação.

**Obs1:** Essa função não precisará conter UI, já que a abordagem não suporta essa camada.

Essa tarefa poderá ser feito seguindo as seguintes orientações:

**1)** Criar uma nova plataforma smartWatch em java:

- platform AndroidSmartWatch: Java { }

**2)** Implementar a função global details *ControllerDetailsGrid* para a plataforma Android.

**Obs2:** Pode copiar e colar essa função do modelo de plataforma *AndroidController,* já que a linguagem e os detalhes técnicos são os mesmos entre as plataformas.

**Obs3**: Para distinguir o nome dos adapters entre o smartwatch e o smartphone, alterar a definição automática do nome da chamada na função global *ControllerDetailsGrid* acrescentando a palavra Watch. Ex: ${E.name}WatchAdapter adapter = new ${E.name}WatchAdapter(${val1}.this, ${list});

3) Criar uma função Global para obter os dados do smartphone. Orientações:

implementsGlobal onDataChanged(dataEvents: DataEventBuffer): void{

'''

for (DataEvent event : dataEvents) { //get data by bloutooth

if (event.getType() == DataEvent.TYPE\_CHANGED) {

DataItem item = event.getDataItem();

if (item.getUri().getPath().compareTo("/count") == 0) {

DataMap dataMap = DataMapItem.fromDataItem(item).getDataMap();

Product val = dataMap.getString(COUNT\_KEY);

if(val != null){

dataListProduct.clear();

dataListProduct.addAll(product);

dataListProduct.deferNotifyDataSetChanged();

adapter.notifyDataSetChanged();

}

}

} else if (event.getType() == DataEvent.TYPE\_DELETED) {

// DataItem deleted

}

}

'''

}

**Obs4:** Para deixar a função global "onDataChanged" genérica, faça sua declaração sem parâmetros. Isso é importante para quando for utilizado em outros smartwatch, como iOS por exemplo.

4) Na GPL você irá criar uma classe (sugestão: MainActivityWatch) para obter os produtos de um smartphone Android em um smartwatch via bluetooth. Orientações:

- Poderá usar na classe "MainActivityWatch" a função global "ControllerDetailsGrid" e a "onDataChanged";

- Caso utilize a função global "ControllerDetailsGrid", deverá configurar as classes genéricas exigidas pela função.

Sugestões de nomes para as classes genéricas: ProductSmartAdapterConf, ProductSmartControllerList e ProductSmartConfMethod; e

- Poderá criar a classe ProductWatchAdapter com as funções globais Adapter e getView para a criação do adapter.