Game UI framework powered by Python

Fear not, there are almost no Tanks!

Ivan Nikolaev
WoT Gameplay Developer
i_nikolaev@wargaming.net

Introducing World of Tanks

…from developer's perspective

Introducing World of Tanks

…from developer's perspective

- MMO action game featuring PvP battles between mid-20th century armored vehicles.
- Complex meta-game.
- Frequent game events: New Year, Steel Hunter, Racing and more.
- Long living project, almost 10 years.

The Good Old Days

2009 (alpha) - 2010 (release)

Today

(It took a while to catch this moment)

Hangar

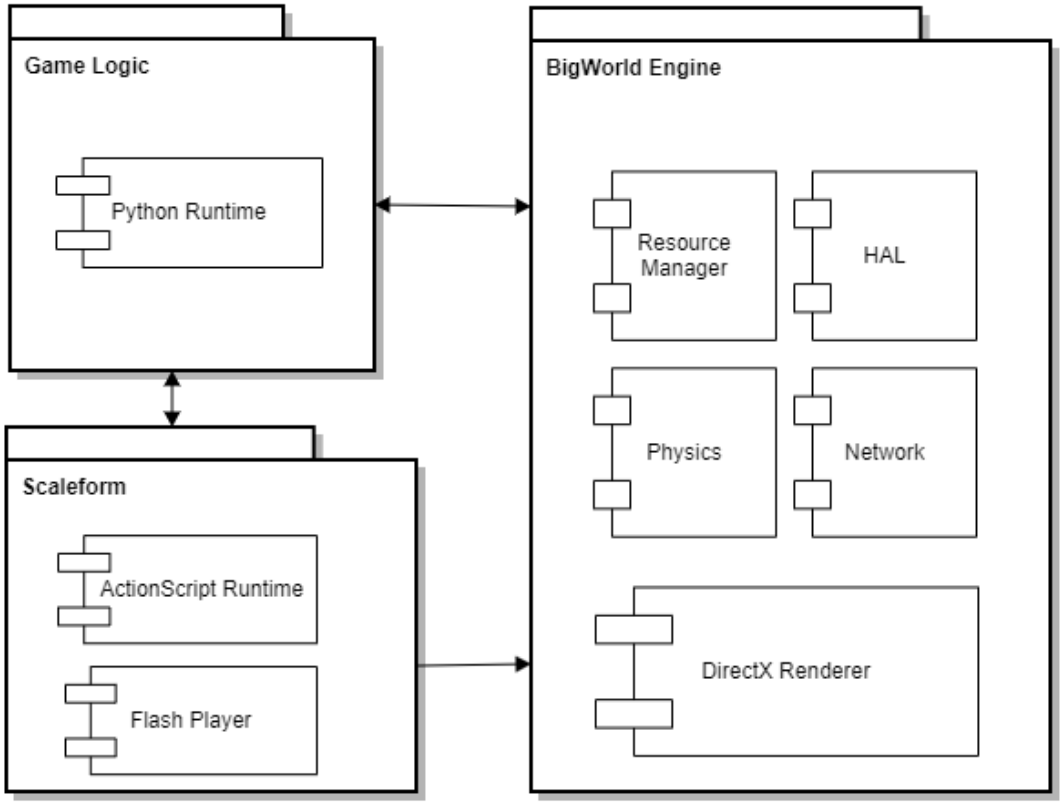
There are simpler office suite interfaces

Hangar exploded

For those who weren't impressed by the previous slide

WoT Client High Level Structure

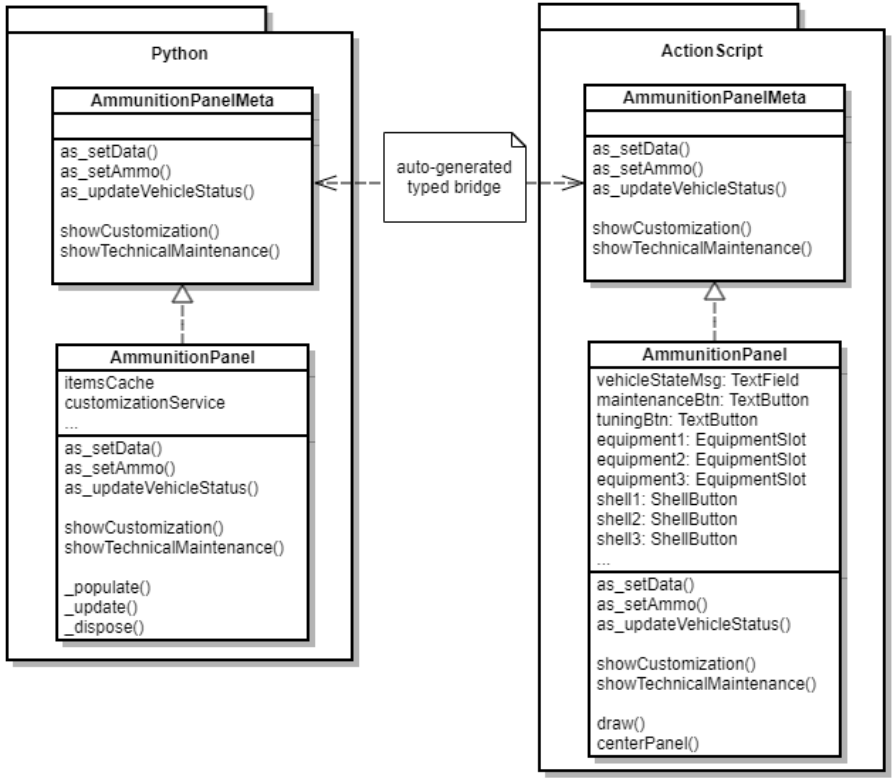
The good old days.



- BigWorld engine for rendering and simulation.
- Flash + ActionScript for UI, integrated with the engine (Scaleform SDK).
- Python 2.7 for game logic scripting.

Example component – Ammunition Panel

Python – Scaleform Direct Access API (Old style, Mirrors framework)



Example component – Ammunition Panel

Python class – some code

```
class AmmunitionPanel(AmmunitionPanelMeta, IGlobalListener):

    def showTechnicalMaintenance(self):
        self.fireEvent(LoadViewEvent(VIEW_ALIAS.TECHNICAL_MAINTENANCE), EVENT_BUS_SCOPE.LOBBY)

    def showCustomization(self):
        self.service.showCustomization()

    def showModuleInfo(self, itemCD):
        if itemCD is not None and int(itemCD) > 0:
            shared_events.showModuleInfo(itemCD, g_currentVehicle.item.descriptor)

    def _populate(self):
        self.startGlobalListening()

    def _dispose(self):
        self.stopGlobalListening()
```

Example component – Ammunition Panel

Python class – some code continued…

```
# …
def _update(self):
    if g_currentVehicle.isPresent():
        vehicle = g_currentVehicle.item

        counter = vehicle.getC11nItemsNoveltyCounter(self.itemsCache.items) \
            if vehicle.isCustomizationEnabled() else 0
        self.as_setCustomizationBtnCounterS(counter)

        counter = AccountSettings.getCounters(BOOSTERS_FOR_CREDITS_SLOT_COUNTER)
        self.as_setBoosterBtnCounterS(counter)

        devices = getFittingSlotsData(vehicle)
        self.as_setDataS({'devices': devices})   # much more stuff here, actually

        shells = getAmmo(vehicle.shells)
        self.as_setAmmoS(shells)

        message = vehicle.getHangarMessage()
        self.as_updateVehicleStatusS({'message': hangarMessage})
```

Problems

- Tight coupling between Python and ActionScript.
- Free-form `dict`-s exchange.
- Redundant draw calls.
- AS programmers are a rare species.
- Sparse state, difficult debugging.
- Out of date and poor tooling.
- Scaleform discontinued since 2018.

First attempt: In-Game Browser

Chromium Embedded Framework

First attempt: In-Game Browser

Chromium Embedded Framework

- Make game UI using HTML5 & JavaScript!
- Open source!
- Modern tools and frameworks (React, Vue).
- Armies of developers available.
- Complex integration (IPC messaging, off-screen rendering).
- Limited use case.
- Performance problems.
- Limited communication with the web app (JS).

Try again: in-house solution

Unbound Framework

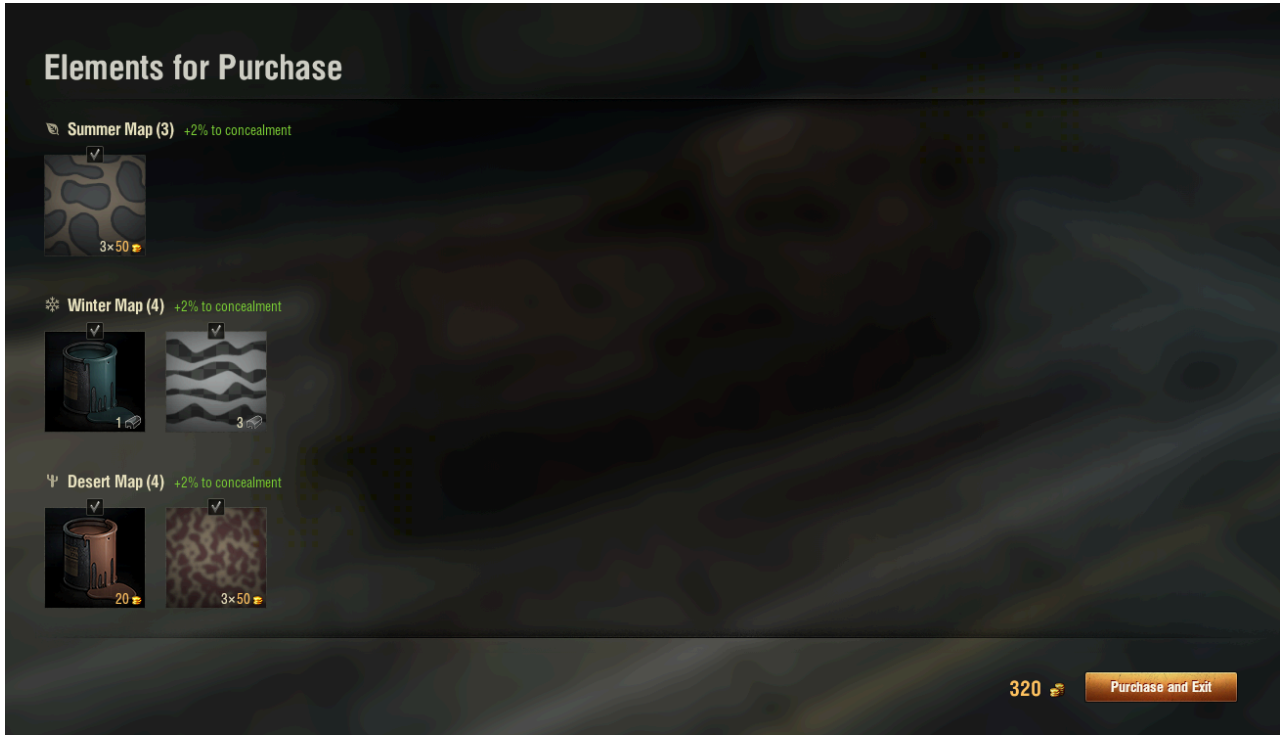
Try again: in-house solution

Unbound Framework

- Custom Lisp-like language ☺
- Good performance.
- Fits well into the existing UI framework.
- Same problems with tooling as ActionScript.
- Same problems with developers availability as ActionScript.
- Dead end.

Hey, web tech was cool!

GameFace Framework



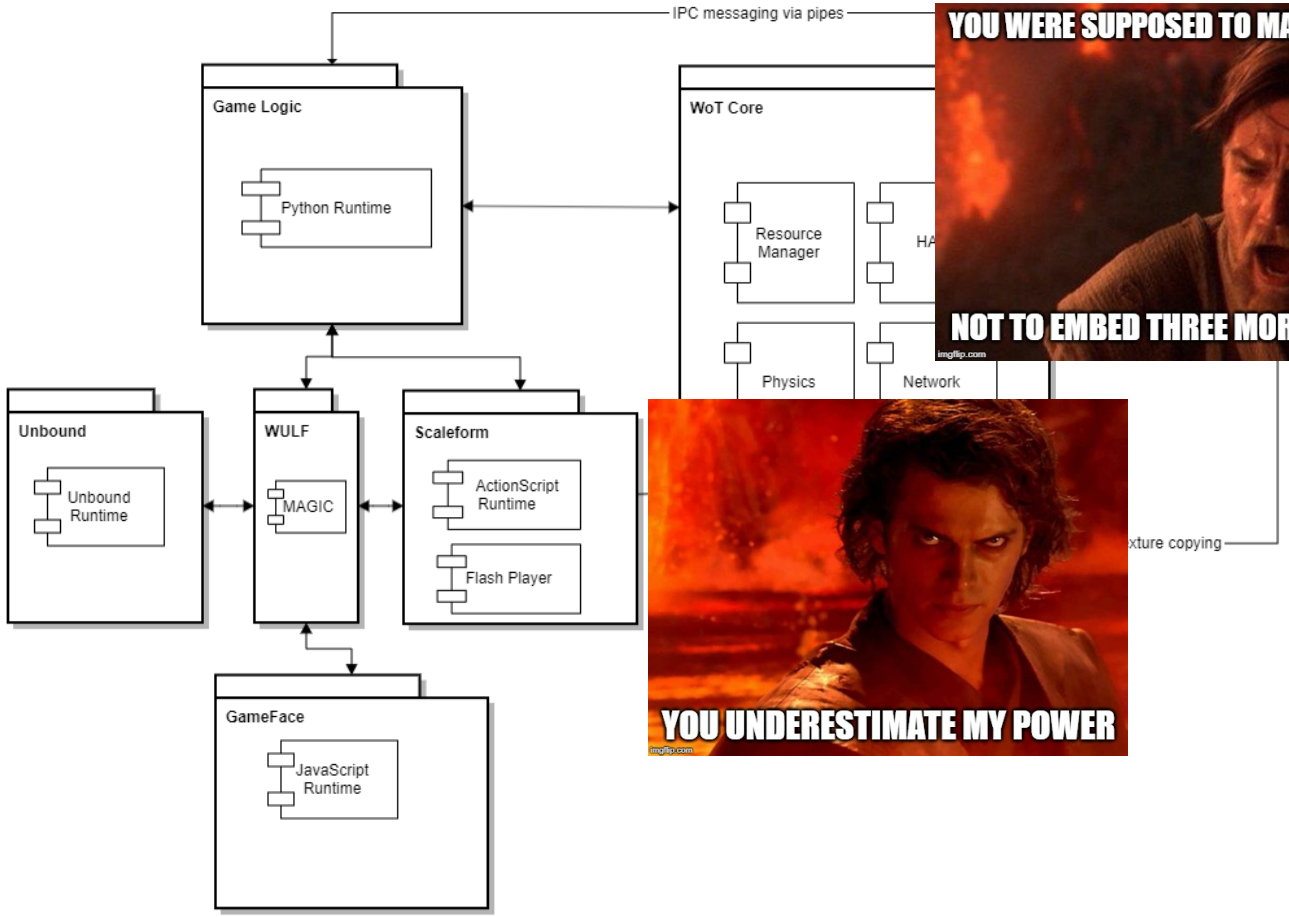
Hey, web tech was cool!

GameFace Framework

- Browser-like runtime.
- Limited HTML/DOM subset.
- Use React, Vue or other JS frameworks and tools.
- Flexible SDK, efficient data binding.
- Does not support DirectX 9.
- Proprietary.

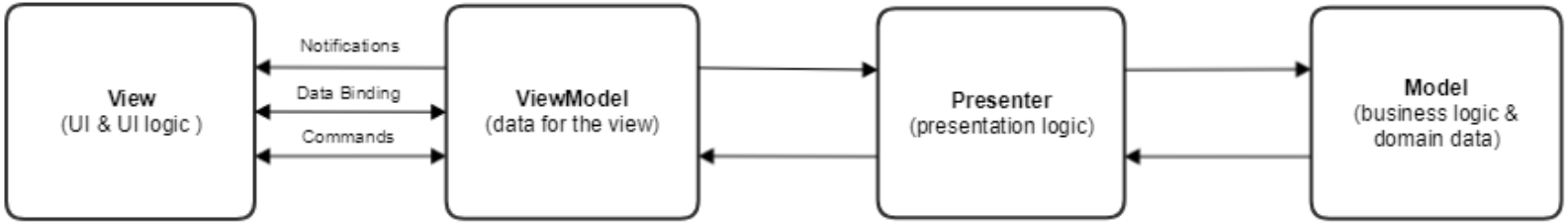
WoT Client High Level Structure

Today



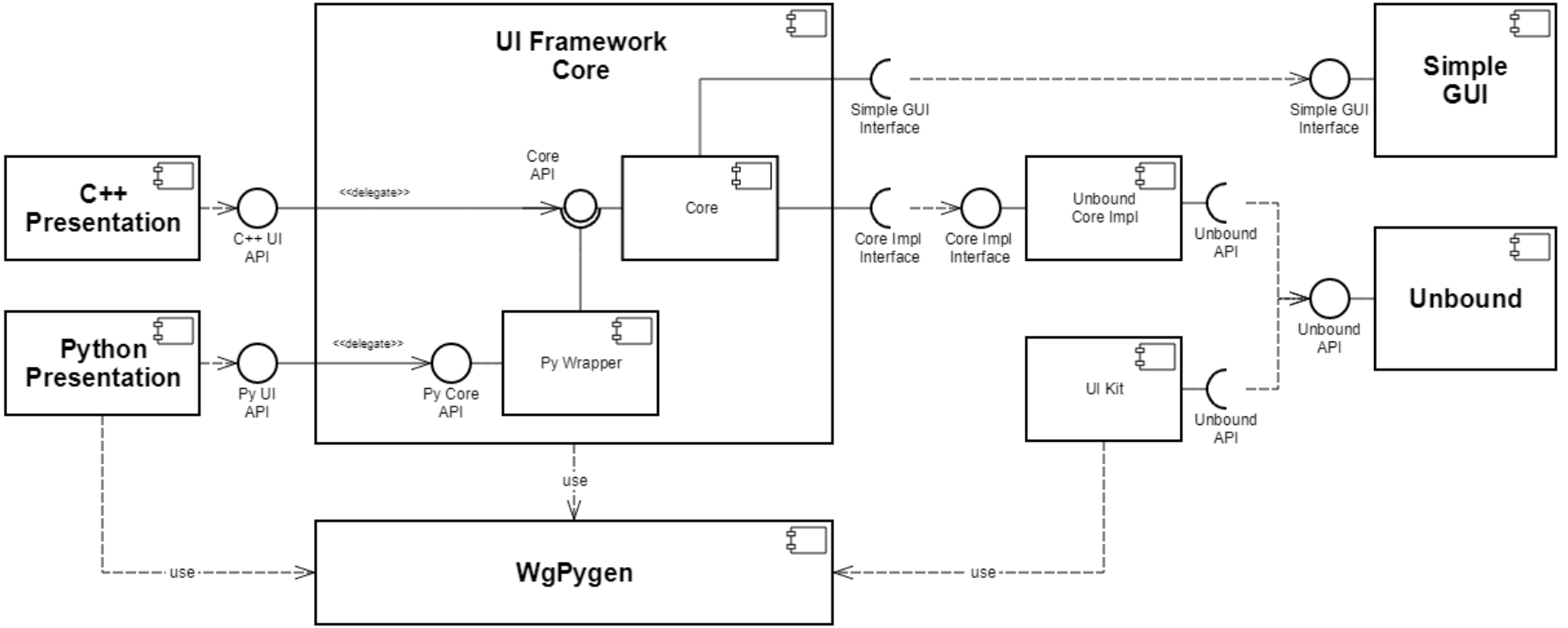
WOT UI LIGHTWEIGHT FRAMEWORK (WULF)

Enter the MVVM Pattern



- View: pure UI.
- ViewModel: typed container with data binding and signals (commands).
- Presenter: bridge to the application.
- Model: actual application data and logic.

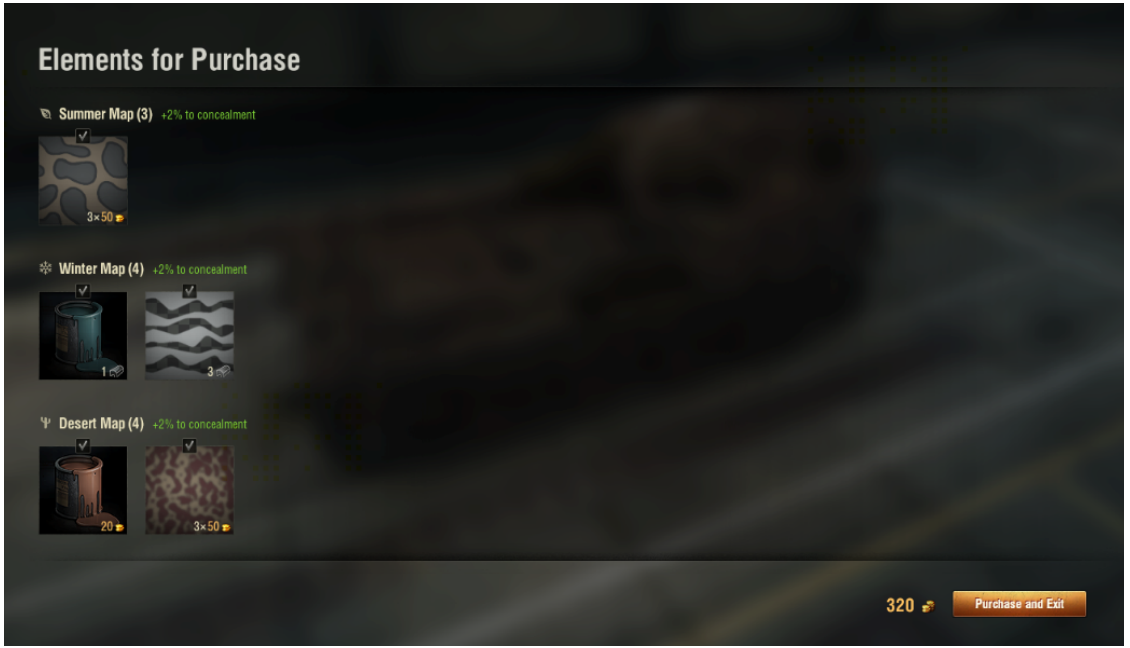
WULF main modules



Example component – Customization Cart

Business requirements

- List the customization entries for purchase for each season.
- Allow for cart change via selection.
- Recompute the price based on selection.
- Initiate the purchase transaction on “Purchase and Exit” press.



Example component – Customization Cart

ViewModel definition – root

```
# cart_model.yaml
name: cart_model

properties:
  - name: isEnoughMoney
    type: bool

submodels:
  - name: seasons
    model: $gui.impl.wrappers.user_list_model.UserListModel
    generic:
      - type: model
        path: views.lobby.customization.cart_season_model

  - name: totalPrice
    model: $gui.impl.wrappers.user_compound_price_model.UserCompoundPriceModel

commands:
   - name: onSelectItem
   - name: onBuyAction
```

Example component – Customization Cart

ViewModel definition – season items

```
# cart_season_model.yaml
name: cart_season_model

properties:
  - name: name
    type: string

  - name: count
    type: number

submodels:
  - name: items
    model: $gui.impl.wrappers.user_list_model.UserListModel
    generic:
      - type: model
        path: views.lobby.customization.cart_slot_model
```

Example component – Customization Cart

ViewModel definition – single item (slot)

```
# cart_slot_model.yaml
name: cart_slot_model

properties:
  - name: id
    type: number

  - name: selected
    type: bool

  - name: icon
    type: string

  - name: quantity
    type: number

submodels:
  - name: price
    model: $gui.impl.wrappers.user_compound_price_model.UserCompoundPriceModel
```

Example component – Customization Cart

ViewModel Python generated class

```
class CartModel(ViewModel):

    @property
    def seasons(self):
        return self._getViewModel(0)

    @property
    def totalPrice(self):
        return self._getViewModel(1)

    def getIsEnoughMoney(self):
        return self._getBool(9)

    def setIsEnoughMoney(self, value):
        self._setBool(9, value)

    def __init__(self):
        self._addViewModelProperty('seasons', UserListModel())
        self._addViewModelProperty('totalPrice', UserCompoundPriceModel())
        self._addBoolProperty('isEnoughMoney', False)
        self.onSelectItem = self._addCommand('onSelectItem')
        self.onBuyAction = self._addCommand('onBuyAction')
```

Example component – Customization Cart

ViewModel TypeScript generated interfaces

```
export interface CartModel extends GFViewModel {
    isEnoughMoney: boolean;
    readonly seasons: UserListModel<CartSeasonModel>;
    readonly totalPrice: UserCompoundPriceModel;
    readonly onSelectItem: (args?: GFCommandArguments) => void;
    readonly onBuyAction: (args?: GFCommandArguments) => void;
}

export interface CartSeasonModel extends GFViewModel {
    name: string;
    count: number;
    readonly items: UserListModel<CartSlotModel>;
}

export interface CartSlotModel extends GFViewModel {
    id: number;
    selected: boolean;
    icon: string;
    quantity: number;
    readonly price: UserCompoundPriceModel;
}
```

Example component – Customization Cart

Python presentation code

```
class CustomizationCart(ViewImpl):

    def _onLoading(self):
        with self.getViewModel().transaction() as model:
            for seasonType in SEASONS_ORDER:
                seasonModel = CartSeasonModel()
                seasonName = SEASON_TYPE_TO_NAME.get(seasonType)
                seasonModel.setName(seasonName)
                seasonModel.setCount(self.__counters[seasonType])
                fillItemsListModel(seasonModel.items, self.__items[seasonType])
                model.seasons.addViewModel(seasonModel)

            self.__updatePrice(model)

    def __updatePrice(self, model):
        price = getTotalPurchaseInfo(self.__items).cart.price
        isEnoughMoney = isTransactionValid(self.__moneyState, price)
        model.totalPrice.assign(price.totalPrice)
        model.setIsEnoughMoney(isEnoughMoney)
```

Example component – Customization Cart

Python presentation code – continued

```
# ... Continued ...
def __updatePrice(self, model):
    price = getTotalPurchaseInfo(self.__items).cart.price
    isEnoughMoney = isTransactionValid(self.__moneyState, price)
    model.totalPrice.assign(price.totalPrice)
    model.setIsEnoughMoney(isEnoughMoney)

def __onSelectItem(self, itemId, selected):
    # ... update the selection in the actual cart ...

    with self.getViewModel().transaction() as model:
        self.__updatePrice(model)
        season = self.__purchaseItems[itemId].season
        seasonModel = model.seasons.getItem(season.seasonId)
        fillItemsListModel(seasonModel.items, season.seasonType)

def __onBuy(self):
    if self.__moneyState is MoneyForPurchase.ENOUGH:
        self.__ctx.applyItems(self.__items)
        self.destroyWindow()
```

What we have now

- The ViewModel is the data exchange protocol.
- Property-level granularity, optimized draw calls.
- UI-frontend agnostic framework, hot-swap possible!
- Flexible View and Presenter code.
- Craft interfaces with modern tools.
- Testable and debuggable.

Take away goodies

- Stay data oriented.
- Keep the state flat.
- Use code generation where possible.
- Do not make assumptions on tech lifetime.
- Design for change.
- Do not rush at center ☺

Thanks!

Questions?

Email: i_nikolaev@wargaming.net
Twitter: V0idExp