# Общая логика

Новая концепция формы - новый способ работы с формой, которая поменяла способ общения между сервером и клиентом, а именно коммуникация между ними происходит в форме запрос/команды. Клиент делает запрос и получает список команд, которых ему надо выполнить.

В новой концепции появилась иерархия форм - теперь есть такие понятия, как родительские и дочерние формы.

Если форма была создана не с другой формы, то она является корневой и не имеет родителя.

Если форма была создана с другой формы (поле БО, нажатие на кнопку добавить), то новая созданная (вложенная) форма будет дочерной, а ее родителем будет предыдущая форма

## Первый этап

Первый этап сохранении/изменении инстанции - это создание черновика.

Черновик создается в регистре **InstanceDraftCreateRegister**.

При создании черновика НАДО передать всю полезную информацию про контекст, откуда запрос делается. Под полезной информацией понимается уровень вложенности текущей инстанции, есть ли вообще текущая инстанция (текущей инстанции может и не быть, пример - нажатие на + над таблицой).

При создании черновика происходят следующие события (очередность сохранена):

Копирование с поля

Формулы

Скрипт на открытие формы

Активация ловушки-липучки (объязует пользователя сохранить данную форму)

## Второй этап

Второй этап сохранении/изменении инстанции - это изменение черновика.

Данные о полей берутся с регистра **InstanceFormRegister**

Эти данные хранят минимальную информацию; остальная информация о полях, включая storedValue, извлекается со следующих регистров:

InstanceFieldFormRegister

InstanceFieldFormDataRegister

Значение записываются в черновик с помощью метода: **InstanceFieldFormRegister.saveBoiFieldValue()**

Алгоритм действий внутри вышенаписанного метода следующая:

Проверка всех BoiFieldValueToSave

Обновление участников

Копирование полей

Вычисление формул

Скрипты на изменение полей

Применение изменений

Валидация формы

## Третий этап

Третий (и последний) этап сохранении/изменении инстанции - это применение/отмена черновика.

При отмене формы отрабатывается скрипт на закрытие формы.

Также при отмене формы удаляются:

Запрашиваемая форма

Прямые потомки данной формы

Непрямые потомки данной формы

Непрямыми потомками формы являются потомки прямых потомков формы (процесс рекурсивный).

Метод, который отвечает за отмену формы: **InstanceDraftRegister.removeDraft()**

При применении форма сначала валидируется. После запускается скрипт на сохранение формы.

Если форма имеет родителя, то применятся она будет в родительский черновик.

Если форма не имеет родителя, то она применяется в саму инстанцию.

Метод, который отвечает за применение формы: **InstanceDraftRegister.validateAndApplyAndRemoveDraft()**

# Команды формы

В новой концепции почти все действия клиента контролируется апи.

Это реализовано с помощью команд.

После валидации формы клиенту будет отправлен объект **BoiFormResultUpdate**, который содержит различные виды команд.

Есть следующие виды команд:

BoiFormCommandUpdate

BoiFieldFormCommandUpdate

BoiFieldFormCommandUpdateSometimes

## BoiFormCommandUpdate

Данный тип команды группирует в себе все команды, которые относятся к самой форме.

Пример: показ кнопок, показ уведомления по форме, закрытие формы.

## BoiFieldFormCommandUpdate

Данные тип команды группирует в себе все команды, которые относятся к полям формы.

Пример: обновление значения поля, показ ошибок у поля.

## BoiFieldFormCommandUpdateSometimes

Данные тип команды группирует в себе все команды, которые относятся к метаданным полей формы.

Пример: скрытие поля, показ поля, изменение объязательности или свойства "Только для чтения".

### Разница между веб командами и мобильными командами

Из за того, что веб версия формы и мобильная версия формы архитектурно отличаются, команды для них тоже отличаются. Однако основая логика у них одинаковая, отличие - в path (путь).

Поэтому было принято решение конвертировать веб команды в мобильные команды вместо написания DRY кода.

Но разработчику не надо во всех местах конвертировать команды - достаточно над методом, который возвращает **BoiFormResultUpdate** написать аннотацию @CommandReturnConvert.

Есть одно ограничение: метод, надо которым пишется данная аннотация, в виде параметра должен принимать **AuthUserInfo** в любом порядке.

Данная логика реализована с помощью AOP.

Сама логика конвертации находится в **InstanceCommandConvertRegister.convertToMobileCommands()**, если добавится новая команда и команда требует другого подхода для мобильной версий, то надо именно сюда написать логику

# Валидация формы

Валидация формы происходит в **InstanceDraftRegister.validateDraft()**.

Есть следующие стратегии валидации формы:

Полная

Измененные поля

Сохранение сигнатуры

Первая валидация

Они отличаются количеством валидируемых полей.

Например, при "первой валидации" поля НЕ валидируются.

При "полной" или "сохранении сигнатуры" валидируются все динамические поля БО.

При "измененных полях" валидируются поля, которые были изменены внутри этого черновика.

Данные стратегии вызываются в определенных жизненных циклах формы.

Так, при создании черновика формы вызывается первичная валидация (важно учесть, что вызов зависит от клиента, то есть если запрос не будет сделан, сам апи не будет вызывать, и данный жизненный цикл может просто "выпасть")

При изменении поля используется стратегия "Измененные поля", и это контролируется апи.

При сохранении формы используется стратегия "Полная", и это контролируется апи.

При подписании формы используется стратегия "Сохранение сигнатуры", и это контролируется отчасти апи. Подписание инстанции не работает в черновике, поэтому перед подписанием надо создать инстанцию (если она существует только в черновике), либо же зафиксировать все изменения. Если клиент вызовет метод **InstanceDraftRegister.createInstanceBySignature()**, апи использует данную стратегию.

## Валидация полей

Поля черновика валидируется с помощью **BoiFieldValidator**.

Процесс валидации разбит на несколько "раундов".

Разделение валидации на несколько раундов (этапов) обосновывается гибкостью, понятностью и масштабируемостью.

Гибкость - раунды можно переопределить для конкретных видов полей

Понятность - раунды логически разделяют процесс валидации, с меньшим объемом кода легче работать

Масштабируемость - можно легко добавлять новые раунды, при этом необъязательно для всех типов полей
явно указывать логику, достаточно указать единую логику в родительском классе

## Раунды

Существуют следующие виды раундов:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Можно переопределить | Объязательность | Очередность |
| Pre-validation | нельзя | объязателен | 1 |
| Parse validation | нельзя | объязателен | 2 |
| Empty validation | можно | объязателен | 3 |
| Value validation | можно | не объязателен | 4 |
| Logic validation | можно | не объязателен | 5 |
| Post-validation | нельзя | объязателен | 6 |

### Pre-validation

На данном этапе значение может быть пустым

Проверяются общие, легкие проверки (например, проверка на объязательность)

### Parse validation

На данном этапе значение не может быть пустым

Значение проверяется на корректность storedValue

Пример: в INPUT\_NUMBER нельзя хранить символы

### Empty validation

На данном этапе значение не может быть пустым

Значение проверяется на пустоту, но не на null и blank

Например, "[]" - не null и не blank, однако это пустое значение для полей типа BO и CO

### Value validation

На данном этапе значение не может быть пустым

Проверяется дополнительная логика storedValue

Например, "Parse validation" не проверяет корректность паттерна телефонного номера

### Logic validation

На данном этапе значение не может быть пустым

Проверяется бизнес-логика

### Post-validation

На данном этапе значение не может быть пустым

Проверяются общие, тяжелые проверки

Например, проверка на уникальность

### Общая логика прохождения раундов

Раунды проходят один за другим, это значит, что неуспешное прохождение одного раунда прерывает весь процесс валидации.

По этой же причине рекомендуется проводить раунды по мере возрастания сложности проверки.

### Рекомендации

Очень желательно, чтобы новые проверки попадали в нужные раунды.

Таким образом проверка будет находиться в логически правильном месте и облегчить жизнь другим разработчикам.

Пример: допустим, что все текстовые поля теперь должны начинаться на "A" (все, без исключения). На первый взгляд можно подумать, что проверка относится к Logic validation, так как проверяется бизнес логика. Однако в данном случае проверка относится к Value validation, так как проверяется значение на какой то паттерн.

Точно так же, новый раунд (помимо этих 6) рекомендуется добавлять, ТОЛЬКО если все 6 раундов не удовлетворяют условия.

Помните, что проверку КАЖДОГО поля можно кастомизировать, а именно переопределить раунды, которые разрешаются (см выше таблицу).