

**Расширение функционала платформы «ИС.ПРОМЕТЕЙ  
V2»: руководство по кастомизации платформы**

2025 г.

## СОДЕРЖАНИЕ

1.	Возможности кастомизации платформы .....	5
2.	Реализация пользовательских настраиваемых действий.....	6
2.1.	Фронтенд часть реализации настраиваемого действия .....	6
2.2.	Бэкенд часть реализации настраиваемого действия .....	9
2.3.	Активация действий в системе.....	16
2.3.	Примеры реализации действий .....	17
3.	Реализация системных действий .....	23
3.1.	Прямая подписка на топик .....	23
3.2.	Реализация пользовательских действий через расширение AbstractServiceAction.....	26
4.	Реализация портлетов .....	28

## Перечень условных обозначений, терминов и сокращений

Таблица 1 – Список принятых обозначений, сокращений / терминов

Сокращение / Термин	Определение
БД	Базы данных
Конструктор БП	Отдельный модуль платформы, используемый для создания и автоматизации бизнес-процессов Системы
Кастомный	Решения, функции или компоненты, созданные или адаптированные в соответствии с конкретными требованиями, выходящими за рамки стандартных возможностей системы или библиотек
Портлет	Подключаемый, сменный компонент пользовательского интерфейса веб-портала
Платформа	Российская low-code платформа «ИС.ПРОМЕТЕЙ V2»
Form.io	Платформа для управления формами и данными, которая позволяет создавать сложные веб-формы, управлять ими и обрабатывать собранные данные
JSON	Открытый стандартный формат файла для обмена данными, в котором для хранения и передачи данных используется удобочитаемый текст. Файлы JSON хранятся с расширением *.json
Maven проект	Проект, использующий Apache Maven для управления сборкой, зависимостями и жизненным циклом портлета

## **Аннотация**

Настоящий документ представляет собой подробное руководство для разработчиков и технических специалистов, занимающихся адаптацией и расширением возможностей платформы ИС.ПРОМЕТЕЙ V2. В документе рассмотрены ключевые аспекты кастомизации платформы, включая создание пользовательских действий, разработку кастомных портлетов.

## **1. Возможности кастомизации платформы**

Low-code платформа ИС.ПРОМЕТЕЙ V2 предоставляет мощный набор стандартных инструментов, таких как predefined действия и портлеты, которые позволяют быстро создавать и внедрять бизнес-приложения без необходимости глубоких знаний в программировании. Однако могут возникать задачи, требующие расширения базового функционала платформы для удовлетворения уникальных требований заказчика.

Данный документ призван предоставить разработчикам и техническим специалистам четкое руководство по расширению функционала платформы, а также показать, как можно эффективно использовать её возможности для создания уникальных решений. В данном документе будут рассмотрены возможности кастомизации платформы, позволяющие создавать кастомные инструменты для расширения возможностей платформы.

В документе будут рассмотрены следующие возможности:

- Создание пользовательских и системных действий (руководство по разработке и интеграции новых функций);
- Разработка кастомных портлетов (инструкции по созданию и настройке проектов портлетов).

## 2. Реализация пользовательских настраиваемых действий

В платформе для перехода между различными этапами бизнес-процесса используются действия (см. рис. 1), которые располагаются в Конструкторе бизнес-процессов, в Редакторе модели. Визуально в модели действие изображено в виде «стрелки», которая осуществляет переход от одного этапа бизнес-процесса к другому.



Рисунок 1 – Визуальное отображение действий в Редакторе модели

Для каждого действия возможно указать его тип. На платформе предоставлен набор стандартных действий, которые можно использовать для перехода между формами или выполнения иной логики. При необходимости для действий возможно реализовать нестандартную логику, расширив функциональность платформы.

Логика выполнения действия в low-code платформе ИС.Прометей разделена на два этапа: фронтенд и бэкенд. Такой подход позволяет гибко управлять процессом выполнения действий, обеспечивая как взаимодействие с пользователем на стороне интерфейса, так и выполнение бизнес-логики на стороне сервера.

### 2.1. Фронтенд часть реализации настраиваемого действия

Разработчик фронтенда может писать JavaScript-код, выполняемый между нажатием кнопки и обращением к бэкенду. Этот код позволяет: валидировать данные формы, выполнять вычисления, информировать пользователя и т.д.

При выполнении фронтенд части действия логика выполняется в текущем контексте, в котором доступна переменная инстанса `form.io formInstance` которую можно использовать для получения/изменения данных формы или валидации, также `applicationAdapter` в которой хранятся данные текущего контекста записи. Они также доступны в консоли браузера на странице выполнения действия для отладки.

#### 2.1.1. Реализаций настроек действия

Для создаваемых действий может потребоваться возможность на пользовательском интерфейсе задавать настройки действия и начальные данные, которые будут использоваться при выполнении действия. Настройки действия задаются аналитиком в Редакторе действий.

В случае, если для действия были реализованы настройки, аналитику необходимо выполнить следующее:

1. Разместите подготовленное разработчиком действие на схеме бизнес модели.
2. Перейдите в действия, выберите форму с этим действием и нажмите на шестеренку - настройка действия.
3. Будут выведены параметры, которые разработчик требует от вас ввести. Обязательные помечены красной звездочкой. А дополнительные находятся в выпадающем списке, их можно добавить после выбора и нажатия на кнопку “Добавить”.
4. Нажмите “Применить” и “Сохранить”, чтобы действие сохранилось. Нажмите “Отменить”, чтобы отменить внесенные изменения.

Разработчику при реализации настраиваемого действия необходимо как обычно создать новое действие, но дополнительно указав флаг `setWithSettings`, а также перечислить определения настроек с помощью метода `setSettingsDefinition`. Об этом далее.

Метод Определение настроек (`settingsDefinition`) возвращает JSON-объект, который описывает настройки для конкретного действия:

- **Значение (value)** – ключ, по которому можно будет получить значение настройки, является обязательным;
- **Наименование (label)** – то, что будет выведено при настройке действия напротив настройки;
- **Условия отображения (conditional)** – необязательный параметр, Js-код, который будет должен возвращать булево значение / любой результат, считающийся в Js эквивалентным `true` (число не равное 0, не пустая строка и т.д) / `false` (число равное 0, пустая строка, `null`, `undefined`). Обращение к значениям настроек осуществляется через `this.settings[ключ настройки]`. По умолчанию настройка всегда выводится;
- **Тип** – тип выводимого инпута в настройках (чекбокс – `checkbox`, выпадающий список – `select`, текст – `text`). По умолчанию настройка имеет тип `text`;
- **Колонка** – косметическая настройка, является необязательной. Поскольку настройки выводятся в 2 колонки, то для указания вывода в первую используется значение `col-1`, во вторую `col-2`. По умолчанию настройка помещается в первую колонку;
- **Дополнительный параметр (additional)** – настройка будет выводиться в выпадающем списке.

Для получения значений, введенных аналитиком, нужно обратиться к геттеру `getSettings`. Из него по ключу, указанному в определении настройки можно достать значение.

### 2.1.2. Функции доступные в контексте выполнения Frontend части действия

Ниже приведен список ключевых функций и объектов, которые обычно доступны в этом контексте:

- `showSuccess('Успешное выполнение действия тест')` – показать success alert;
- `showError('Ошибка тест')` – показать error alert;
- `resetApplication()` – перезагрузить данные заявки и форму.

### 2.1.3. Схема выполнения действия на фронтонной части

Ниже представлена схема выполнения действия на фронтонной части (см. рис. 2).

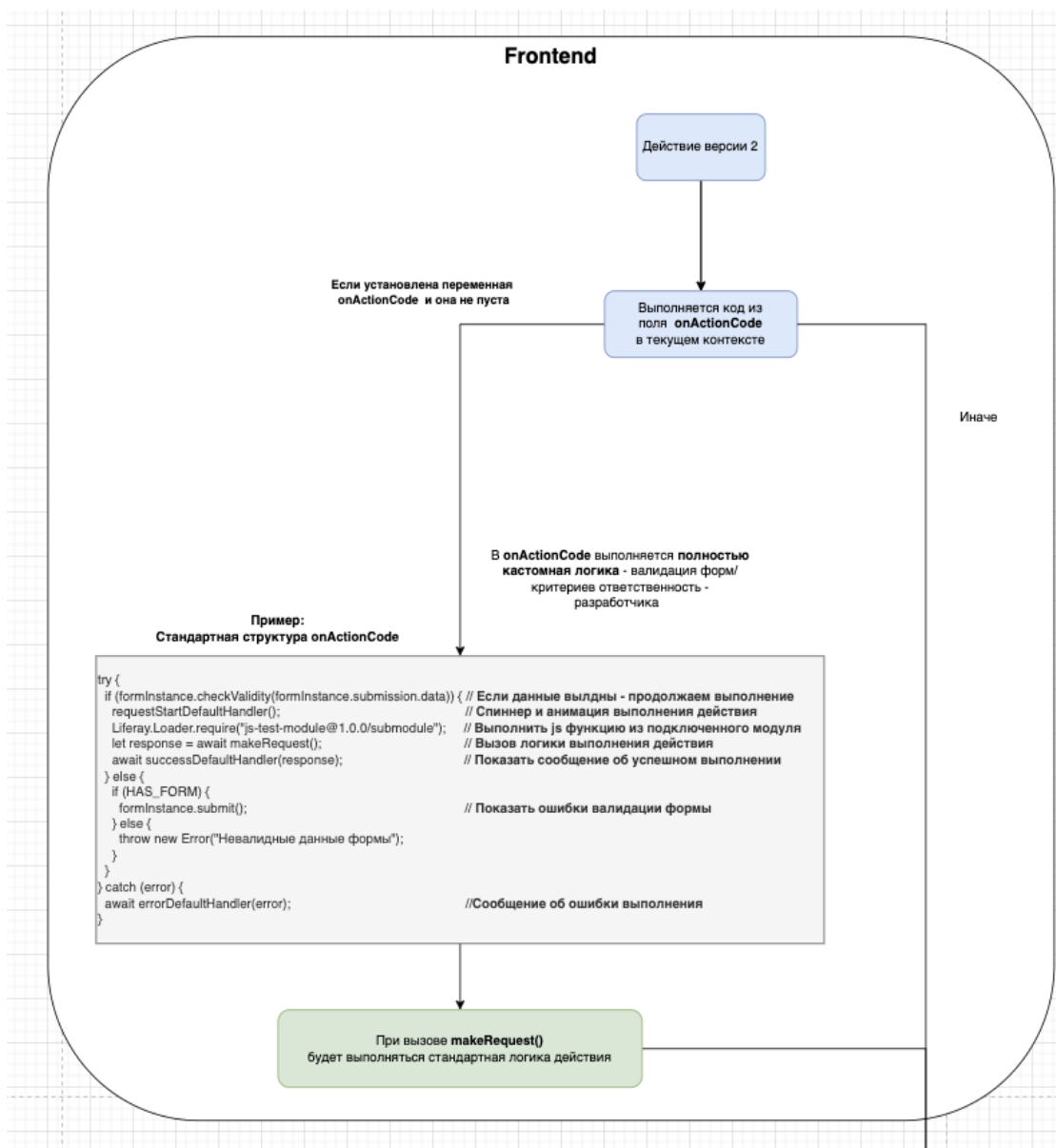


Рисунок 2 – Схема выполнения действия на фронтонной части

## 2.2. Бэкенд часть реализации настраиваемого действия

Эта часть логики выполняется после завершения фронтенд-этапа. Код бэкенда написан на языке Java, и для его реализации используется базовый класс `AbstractBaseAction`, от которого необходимо наследоваться. Этот класс содержит основную логику выполнения действия, которую можно расширять и адаптировать под конкретные задачи.

### 2.2.1. Настройка действия

При создании нового действия в его конструкторе требуется задать несколько обязательных полей:

- **name** (Уникальный ключ действия) – уникальный идентификатор действия, например, "CustomAction";
- **actionType** (Тип действия) – тип действия, который определяет его поведение. Например, `ActionType.FORM`;
- **version** (Версия действия) – Версия действия, например, 2L;
- **actionTag** (Группировка действий на интерфейсе) – ключ, по которому действия будут сгруппированы в интерфейсе. Например, "Новые действия";
- **onActionCode** (JavaScript-код для фронтенд-части) – код, который выполняется на этапе фронтенда перед вызовом бэкенда. Пример кода приведен в полном примере ниже;
- **label** (Название действия, отображаемое пользователю) – Название действия, которое будет показано в интерфейсе, например, "Новое действие".

Эти поля необходимы для корректной настройки и интеграции действия в платформу, обеспечивая его уникальность, управляемость и понятность для пользователей.

Пример реализации:

```
public class NewAction extends AbstractBaseAction {

public NewAction() {
    setName("NewActionKey")
        .setActionType(ActionType.FORM)
        .setActionTag("Новые действия")
        .setVersion(2L)
        .setOnActionCode(
            "try {\n" +
            "    if (formInstance.checkValidity(formInstan
ce.submission.data)) {\n" +
            "        requestStartDefaultHandler();\n" +
            "        let response = await makeRequest();\n" +
```

```

        "        await
successDefaultHandler(response);\n" +
        "    }\n" +
        "    else {\n" +
        "        if (HAS_FORM) {\n" +
        "            formInstance.submit();\n" +
        "        }\n" +
        "        else {\n" +
        "            throw new Error(\"Невалидные данные
формы\");\n" +
        "        }\n" +
        "    }\n" +
        "}\n" +
        "catch (error) {\n" +
        "    await errorHandler(error);\n" +
        "}"
    )
    .setLabel("Новое действие");
}
}

```

Перечень необязательных параметров:

- **readonlyDisplay** (Отображение действия на неактивной форме) – принимает значения true/false;
- **backUrlAction** (Переход в реестр после завершения) – принимает значения true/false
- **hidden** (Скрытый) – принимает значения true/false;
- **noRequest** (Не отправлять запрос на сервер для сохранения заявки) – принимает значения true/false;
- **withSettings** (Указание на наличие настроек у действия) – принимает значения true/false;
- **settings** (Объект настроек. Все настройки отображаются в меню "настройки действия" в конструкторе бизнес-процессов);
- **defaultSettings** (Объект настроек по умолчанию).

### 2.2.2. Реализация кастомной логики

Для реализации нестандартной логики необходимо переопределить метод:

```
public CamundaActionResponse invoke(Map<String, Object> params)
```

В контексте выполнения метода **invoke** (класс **AbstractBaseAction**) для завершения выполнения действия следует вызвать метод **complete(params)**, который переведет запись на

следующую форму. Также доступен метод **long appId = save(params)**;, который сохраняет данные и статусы.

Данные, передаваемые в объекте **Map<String, Object> params**, включают:

- **PARAMS\_HANDLER\_ACTION** – имя действия, для установки имени текущего действия используется метод setName (например, "ExampleDefaultAction");
- **PARAMS\_HANDLER\_DATA** – JSONObject с данными записи;
- **PARAMS\_HANDLER\_OWNER\_ID** – UserId владельца заявки;
- **PARAMS\_HANDLER\_MODEL\_ID** – Id модели бизнес-процесса;
- **PARAMS\_HANDLER\_APPLICATION\_ID** – Id заявки;
- **PARAMS\_HANDLER\_FORM\_ID** – Id формы модели;
- **PARAMS\_HANDLER\_TYPE** – тип заявки (перечисление enums.ApplicationType):
  - 0 – "Неизвестный",
  - 1 – "Личное обращение",
  - 2 – "Обращение через посредника",
  - 3 – "Внешний источник".
- **PARAMS\_HANDLER\_STATE** – статус заявки (StatusModel -> application.status);
- **PARAMS\_HANDLER\_STATE\_STRING** – статус в читаемом виде (например, "Черновик");
- **PARAMS\_HANDLER\_USER** – UserId пользователя, выполнившего действие;
- **PARAMS\_HANDLER\_APPLICATION\_START** – дата начала заявки;
- **PARAMS\_HANDLER\_ORG\_BINDING** – ФИАС-код.

Дополнительные данные, если это обработка внешнего запроса:

- **PARAMS\_HANDLER\_INCOMING\_REQUEST\_DATA** – JSONObject с входящими данными от посредника;
- **PARAMS\_HANDLER\_ORDER\_ID** – номер заявки внешней системы (например, ЕПГУ);
- **PARAMS\_HANDLER\_REQUEST** – название запроса (например, Application/Cancel);
- **PARAMS\_HANDLER\_REQUEST\_APP\_ID** – Id запроса в посреднике (ru.isands.intermed.model.Application);
- **PARAMS\_HANDLER\_EXT\_DATA** – JSONObject с входящими полями.

### 2.2.3. Вспомогательный класс для работы с данными бизнес-процессов

Существует класс **PrometheusSupportUtil** который содержит методы для общего использования, например для поиска записей или изменения статусов.

Список методов класса следующий:

- **createApplication** – позволяет создать новую запись;
- **esSearchApplications** – поиск записей по критериям (будет рассмотрен подробнее ниже);
- **fetchPersistedApplications** – вытягивание полных данных записей по поисковому ответу;
- **esSearchApplicationsFetch** – поиск записей по критериям, возвращает полные данные найденных сущностей;
- **deleteApplication** – удалить запись по id;
- **extractProcessModel** – извлечь модель BPMNModel по переданным данным БП;
- **extractFormModel** – извлечь модель формы FormIOForm по переданным данным;
- **extractTemplateModel** – извлечь модель полей записи FormIOForm по переданным данным;
- **moveApplication** – переместить запись на указанную форму;
- **addUsersToModel** – добавление пользователей к модели БП (участников – переопределение базовой логики);
- **deleteUsersFromModel** – удаление пользователей из модели БП по переданному списку;
- **clearModelUsers** – очистка всего списка участников модели БП;
- **getUsersForModel** – получить список участников для модели;
- **findRequests** – получить список запросов из внешних систем по критериям;
- **resendRequest** – перенаправить запрос во внешнюю систему;
- **requestMapping** – получить маппинг полей запроса/ответа;
- **extractTemplateDefaultValues** – получить список стандартных значений компонентов полей записи;
- **extractFormDefaultValues** – получить список стандартных значений компонентов формы;
- **enrichWithDefaultData** – добавить в данные стандартные значения;
- **changeApplicationState** – изменить статус записи (для сервисного действия);
- **findExternalApplicationId** – получить id внешней записи по id внутренней записи.

Метод **esSearchApplications** позволяет искать записи по заданным критериям. Метод содержит следующие параметры:

- **processIds** – ids бизнес процессов для поиска, должно быть заполнено;
- **criterion** – критерии поиска;
- **size** – размер страницы результата;
- **offset** – сдвиг;
- **includeJson** – включить в результат полный JSON заявки, иначе будут возвращены только индексированные поля;
- **sortField** – поле сортировки, по умолчанию id;
- **sortDescending** – направление сортировки.

В методе **esSearchApplications** параметр **criterion** – это комплексный объект критериев поиска. Объект может состоять из сгруппированных критериев из списка ниже:

- **TermCriterion** – полное совпадение:  

```
test – TermCriterion<String> termCriterion = new  
TermCriterion<>("documents.textField", "test");
```
- **WildcardCriterion** – частичное совпадение:  

```
*tes* – WildcardCriterion wildcardCriterion = new  
WildcardCriterion("documents.textField", "tes");
```
- **TermsCriterion** – совпадение по массиву:  

```
test,test2 – TermsCriterion<String> termsCriterion = new  
TermsCriterion<>("documents.textField", List.of("test", "test2"));
```
- Совпадение **MatchCriterion** – то же, что и **TermCriterion**, за исключением того, что анализирует текст (разбивает на слова, приводит к нижнему регистру);
- **MatchPhraseCriterion** – ищет точную фразу, сохраняя порядок слов;
- **MultiMatchCriterion** – используется для поиска по нескольким полям одновременно. Он работает аналогично **match**, но вместо одного поля принимает список полей;
- **PrefixCriterion** – используется для поиска документов, где поле начинается с указанного префикса, не использует анализатор (анализатор работает только с **match** и **term**);
- **QueryStringCriterion** – поисковая строка:

```
QueryStringCriterion queryStringCriterion = new
QueryStringCriterion(List.of(), "session_user_id:35738 AND
params_handler_application:105838"
```

Условия между собой могут быть скомпонованы при помощи:

- Условия «И»

```
AndComposite andComposite = new AndComposite(List.of(termCriterion,
wildcardCriterion));
```

- Условие «ИЛИ»

```
OrComposite andComposite = new OrComposite(List.of(termCriterion,
wildcardCriterion));
```

Поиск работает по **общим полям** либо по компонентам на форме у которых **установлен параметр** «Использовать при поиске | Индексируемый компонент» то есть такие поля попадают в поисковый движок и по ним доступен поиск.

Список общих полей (начинаются с решетки #):

- **"#archived"** "type": "boolean" – true/false;
- **"#created\_exec\_type"**: "type": "keyword" – TEST/MAIN исполнение БП - Тестовый/Основной;
- **"#ended\_at"**: "type": "date" – дата завершения записи;
- **"#fias\_region"**: "type": "keyword" – orgId id организации создавшей запись;
- **"#id"**: "type": "long" – id записи;
- **"#modified\_at"**: "type": "date" – дата изменения;
- **"#order\_id"**: "type": "keyword" – id записи внешней системы;
- **"#owner\_id"**: "type": "long" – id владельца записи;
- **"#process\_id"**: "type": "long" – id БП – 123;
- **"#process\_key"**: "type": "keyword" – полный первичный ключ БП - 123\_v1;
- **"#process\_version"**: "type": "long" – версия БП – 1;
- **"#role\_participants"**: "type": "long" – ids ролей работавших с записью (имеющих доступ в записи);
- **"#started\_at"**: "type": "date" – дата создания;
- **"#status\_id"**: "type": "long" – id статуса;
- **"#type"**: "type": "long" – тип записи;
- **"#user\_participants"**: "type": "long" – список пользователей, работавших с записью;

- **#active (boolean)** – активна ли запись;
- **#current\_form (long)** – Id текущей формы;
- **#instance\_id (String)** – Instance Id записи на сервере Camunda;
- **#business\_key (String)** – Business key на сервере Camunda;
- **#epgu\_status (long)** – статус ЕПГУ;
- **#epgu\_org (boolean)** – является ли статус ЕПГУ ведомственным или техническим;
- **#source\_type (long)** – тип источника;
- **#root\_application\_id (long)** – Id главной записи подпроцесса.

#### 2.2.4. Схема выполнения действия на бэкенд части

Ниже представлена схема выполнения действия на бэкенд части (см. рис. 3).

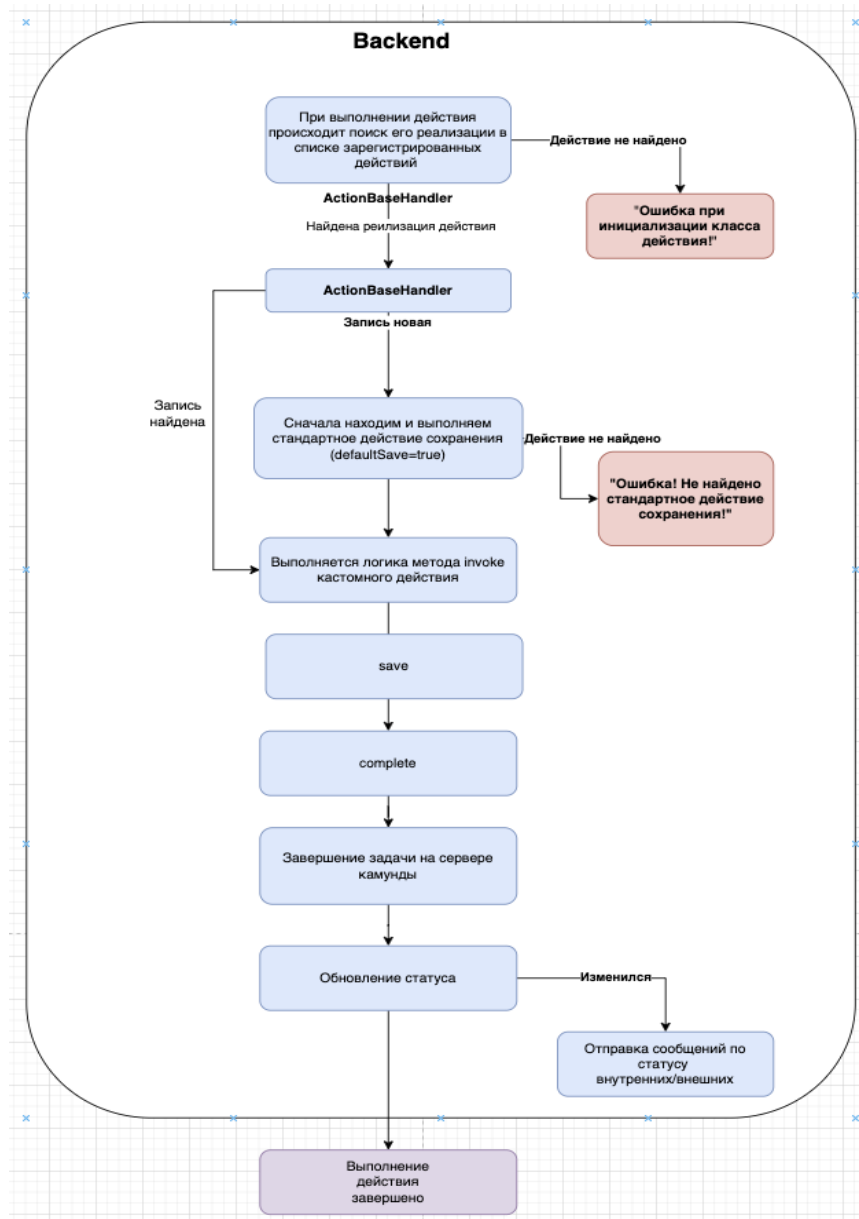


Рисунок 3 – Схема выполнения действия на бэкенд части

### 2.3. Активация действий в системе

Для активации действий в системе (настраиваемых и системный расширяющих **AbstractServiceAction**) нужно добавить класс активатор со списком классов действий, который при старте модуля - добавляет новые действий с список доступных, при деактивации - удаляет.

Пример:

```
@Component(immediate = true)
public class ExampleActionActivator {
    private static final Log log =
LogFactoryUtil.getLog(ExampleActionActivator.class.getName());

    @Reference ActionBaseHandler _service;

    private static Class[] actions = {
        ExampleDefaultAction.class,
        AssignByCustomRolesAction.class,
        CompleteExpertiseAction.class,
        CreateApplicationOnFullPercentAction.class,
        StartExpertiseAction.class,
        UpdateExpertiseResultAction.class,
        ExampleDefaultJSActionV2.class,
        ExampleDefault2JSActionV2.class,
    };

    @Activate
    void activate() {
        log.debug("ExampleActionActivator Example Actions...");
        _service.registerAction(actions);
        log.debug("Done.");
    }

    @Deactivate
    void deactivate() {
        _service.unregisterAction(actions);
    }
}
```

Чтобы классы из данного материала были доступны в модуле нужно добавить зависимости:

```
<dependency>
    <groupId>ru.isands.camunda.core</groupId>
    <artifactId>camunda-template-lib</artifactId>
    <version>${prometheus-version}</version>
    <scope>provided</scope>
</dependency>
```

```

<dependency>
  <groupId>ru.isands.camunda.core</groupId>
  <artifactId>camunda-service-layer-api</artifactId>
  <version>${prometheus-version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>ru.isands.camunda.core</groupId>
  <artifactId>camunda-template-lib</artifactId>
  <version>${prometheus-version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>ru.isands.camunda.core</groupId>
  <artifactId>camunda-service-layer-api</artifactId>
  <version>${prometheus-version}</version>
  <scope>provided</scope>
</dependency>

```

### 2.3. Примеры реализации действий

Ниже представлены примеры реализации действий.

Реализация действия, которое позволяет сохранять данные записи и осуществляет переход на следующую форму:

```

public class InternalDefaultAction extends AbstractBaseAction {

public InternalDefaultAction() {
    setName("InternalDefaultAction")
        .setActionType(ActionType.FORM)
        .setActionTag("Действия платформы V2")
        .setVersion(2L)
        .setOnActionCode(
            "try {\n" +
                "    if
(formInstance.checkValidity(formInstance.submission.data)) {\n" +
                "        requestStartDefaultHandler();\n" +
                "        let response = await makeRequest();\n" +
                "        await
successDefaultHandler(response);\n" +
                "    }\n" +
                "    else {\n" +
                "        if (HAS_FORM) {\n" +
                "            formInstance.submit();\n" +
                "        }\n" +
                "        else {\n" +
                "            throw new Error(\"Невалидные данные
формы\");\n" +
                "        }\n" +
            "    }\n" +
        )
    }
}

```

```

        "    }\n" +
        "}\n" +
        "catch (error) {\n" +
        "    await errorHandler(error);\n" +
        "}"
    )
    .setLabel("[v2] Системное - Стандартное действие");
}
}

```

Реализация действий без проверки валидности данных форме на фронтенд части:

```

public class InternalDefaultNoReqAction extends AbstractBaseAction {

public InternalDefaultNoReqAction() {
    setName("InternalDefaultNotRequiredAction")
        .setActionType(ActionType.FORM)
        .setActionTag("Действия платформы V2")
        .setVersion(2L)
        .setOnActionCode("try {\n" +
            "    requestStartDefaultHandler();\n" +
            "    let response = await makeRequest();\n" +
            "    await successDefaultHandler(response);\n" +
            "}\n" +
            "catch (error) {\n" +
            "    await errorHandler(error);\n" +
            "}")
        .setLabel("[v2] Системное - Действие без проверки");
    }
}

```

Действие открытия записей с применением настроек, поисковых критериев и модификации логики фронтенд-части, которая обрабатывает ответ от бэкенда:

```

public class OpenApplicationAction extends AbstractBaseAction {

    private static final Log log =
LogFactoryUtil.getLog(OpenApplicationAction.class);

    private static final String BUSINESS_MODEL_API_KEY = "bpModel";
    private static final String SEARCH_API_KEY = "searchApi";
    private static final String APPLICATION_COUNT = "applicationCount";
    private static final long MAX_APPLICATION_COUNT = 10;

public OpenApplicationAction() {
    setName("OpenApplicationAction")
        .setActionType(ActionType.FORM)
        .setActionTag("Действия платформы V2")
        .setLabel("[v2] Системное - Открыть запись")
    }
}

```

```

        .setVersion(2L)
        .setWithSettings()
        .setSettingsSchema(loadSettingsDefinition())
        .setExecutingWithoutSave()
        .setOnActionCode("let response = await makeRequest();\n"
            + "    if (response.data.payload_response.appIds) {\n"
            + "        let url = window.location.href;\n"
            + "        url =
url.replace(/&_ru_isands_camunda_application_business_model=\\d+_v\\d+/,
\" \");\n"
            + "        let appIds =
JSON.parse(response.data.payload_response.appIds);\n"
            + "        if (appIds.length > 0) {\n"
            + "            appIds.forEach(id => {\n"
            + "                let replaceStr = \"app_id=\" + id;\n"
            + "                let newUrl = url.replace(/app_id=(-)?\\d+/,
replaceStr);\n"
            + "                window.open(newUrl, \"_blank\");\n"
            + "            });\n"
            + "        }\n"
            + "        else {\n"
            + "            showError(\"Невозможно открыть запись! Записи по бизнес-
процессу не найдены\");\n"
            + "        }\n"
            + "    }\n");
    }

```

```

@Override
public CamundaActionResponse invoke(Map<String, Object> params) {
    log.debug("Performing OpenApplicationAction");
    CamundaActionResponse response = new CamundaActionResponse();
    var settings = getSettings();
    validateSettings(settings);
    List<Long> appIds = handle(settings, params);
    JSONObject responseObject = new JSONObject();
    responseObject.put("appIds", appIds.toString());
    response.setResponse(responseObject);
    log.debug("OpenApplicationAction complete");
    return response;
}

private List<Long> handle(ActionSettings settings, Map<String, Object> params)
{
    JSONObject appdata = new
JSONObject(String.valueOf(params.get(PARAMS_HANDLER_DATA)));
    String modelId = settings.optString(BUSINESS_MODEL_API_KEY);
    long currentAppId =
Long.parseLong(String.valueOf(params.get(PARAMS_HANDLER_APPLICATION_ID)));
    String searchKey = getResultSearchKey(settings);

```

```

        String value =
String.valueOf(appdata.get(settings.optString(SEARCH_API_KEY)));
        MatchCriterion<String> componentCriterion = new MatchCriterion<>(searchKey,
value);
        ApplicationSearchRequest searchRequest = new
ApplicationSearchRequest.Builder()
            .setSortField("id")
            .setBusinessModelIds(List.of(modelId))
            .setCriteria(componentCriterion)
            .build();
        List<ApplicationDTO> applicationDTOS =
ApplicationRemoteServiceUtil.search(searchRequest)
            .getApplications();
        if (!applicationDTOS.isEmpty()) {
            long appCount = getApplicationCount(settings);
            log.debug(String
                .format("Found %d applications, max amount is %d",
applicationDTOS.size(), appCount));
            return applicationDTOS.stream().filter(dto -> dto.getId() != currentAppId)
                .map(ApplicationDTO::getId).limit(appCount).collect(Collectors.toList()
);
        }
        log.debug("Applications not found");
        return Collections.emptyList();
    }

    private String getResultSearchKey(ActionSettings settings) {
        StringBuilder searchKey = new
StringBuilder(settings.optString(SEARCH_API_KEY));
        Optional<BusinessModelDTO> currentModelDTO =
BusinessModelRemoteServiceUtil.getById(settings.optString(BUSINESS_MODEL_
API_KEY));
        if (currentModelDTO.isPresent()) {
            List<FormIOComponent> components = FormIOForm
                .of(currentModelDTO.get().getFormTemplate().getFormDefinition())
                .orElseThrow()
                .extractComponentsByTypes(
                    Arrays.stream(FormIOComponentType.values()).collect(Collectors.toSe
t()));
            components.forEach(FormIOComponent::enrichParents);
            FormIOComponent parentComponent = components.stream()
                .filter(c -> c.getKey().equalsIgnoreCase(searchKey.toString()))
                .findFirst()
                .orElseThrow()
                .parent();
            while (Objects.nonNull(parentComponent)) {
                searchKey.insert(0, parentComponent.getKey() + ".");
                parentComponent = parentComponent.parent();
            }
        }
    }

```

```

    }
    return searchKey.toString();
}

private void validateSettings(ActionSettings settings) {
    log.debug("Validating settings of OpenApplicationAction");
    if (!settings.has(BUSINESS_MODEL_API_KEY) ||
settings.optString(BUSINESS_MODEL_API_KEY)
        .isEmpty()) {
        throw new IllegalArgumentException(
            "Действие настроено неправильно. Не выбран бизнес-процесс");
    }

    if (!settings.has(SEARCH_API_KEY) ||
settings.optString(SEARCH_API_KEY).isEmpty()) {
        throw new IllegalArgumentException(
            "Действие настроено неправильно. Не указан ключ поля для поиска");
    }

    if (settings.has(APPLICATION_COUNT) &&
!settings.optString(APPLICATION_COUNT).isEmpty()) {
        String count = settings.optString(APPLICATION_COUNT);
        if (!StringUtils.isNumeric(count)) {
            throw new IllegalArgumentException(
                " Действие настроено неправильно. Поле \"Количество отображаемых
записей\" должно быть числовым значением");
        }

        if (Long.parseLong(count) <= 0) {
            throw new IllegalArgumentException(
                " Действие настроено неправильно. Поле \"Количество отображаемых
записей\" должно быть больше 0");
        }
    }
    log.debug("Settings validation complete");
}

private long getApplicationCount(ActionSettings settings) {
    long count =
        settings.has(APPLICATION_COUNT) &&
!settings.optString(APPLICATION_COUNT).isEmpty() ?
        Long.parseLong(settings.optString(APPLICATION_COUNT)) :
MAX_APPLICATION_COUNT;
    if (count > MAX_APPLICATION_COUNT) {
        count = MAX_APPLICATION_COUNT;
    }
    return count;
}

```

```

protected ActionSettingsSchema loadSettingsDefinition() {
    return new ActionSettingsSchema.Builder()
        .select()
        .key(BUSINESS_MODEL_API_KEY)
        .required()
        .options(getBusinessModelsList())
        .label("Выберите бизнес-процесс")
        .build()
        .textField()
        .key(SEARCH_API_KEY)
        .label("Укажите ключ индексируемого компонента")
        .required()
        .build()
        .textField()
        .key(APPLICATION_COUNT)
        .label("Количество отображаемых записей")
        .numeric()
        .rule("return ( v <= 10 ) || \"Количество записей не должно превышать
10\"")
        .rule("return ( !v || v > 0 ) || \"Количество записей должно быть больше
0\"")
        .build()
        .build();
}

private List<Pair<String, String>> getBusinessModelsList() {
    List<BusinessModel> models =
BusinessModelLocalServiceUtil.getBusinessModels(-1, -1);
    return models.stream().map(model -> Pair.of(model.pk(),
        String.format("%d - в. %d | %s", model.getId(), model.getVersion(),
model.getName())))
        .collect(
            Collectors.toList());
}
}

```

### 3. Реализация системных действий

В платформе существует тип пользовательской экранной формы «Системная форма», которая располагается (см. рис. 4), которые располагаются в Конструкторе бизнес-процессов, в Редакторе модели. «Системная форма» представляет собой действия, осуществляемые системой без участия пользователя.

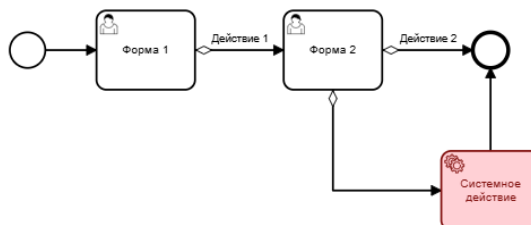


Рисунок 4 – Визуальное отображение системных действий в Редакторе модели

Для системных действий также возможно задавать кастомную логику при необходимости.

#### 3.1. Прямая подписка на топик

В таком случае реализуется ключ, по которому система знает, к какому обработчику отправить на выполнение задачу. Подписчик напрямую регистрируется на конкретный топик (канал).

Реализация:

```
/**
 * Класс делегированной задачи
 * Создается сервис который подписывается на топик
 * По аналогии с messageBus получает сообщение - которое обрабатывает
 *
 */
@Component(immediate = true, service = ExternalSimpleTask.class)

public class ExternalSimpleTask {
    private static final Log log =
    LogFactoryUtil.getLog(ExternalSimpleTask.class.getName());

    // Топик подписчика - должен быть уникальный

    private static final String TOPIC = "ExternalSimpleTaskTopic1";

    ExternalTaskClient client = null;

    @Reference
    ApplicationLocalService _appService;
```

```

@Reference
BusinessModelLocalService _modelService;

public ExternalSimpleTask() {
    init();
}

@Activate
private void activate() {
    log.info("Activated ExternalSimpleTask subscriber");
}

private void init() {

    //Клиент подключения в серверу camunda
    client = ExternalTaskClient.create()
        .baseUrl(SystemUtil.getCamundaServerUrl())
        .asyncResponseTimeout(1000)
        .build();

    //Подписываемся на топик
    client.subscribe(TOPIC
        .handler((externalTask, externalTaskService) -> {

            //Получить все доступные переменные текущей заявки(На сервере
            камунды)

            //Так же системные переменные необходимы для бизнес процесса
            Map<String, Object> vars = externalTask.getAllVariables();

            //Получение сущности заявки
            Application application =
            _appService.findByBusinessKey(externalTask.getBusinessKey());

            //Все UID заявки, в формате uid - EntityType
            Map<String, Object> allUids = application.getAllIds();

            //Получить поле заявки
            //Возможные приведения типа
            (long),(float),(String),(Date),(boolean)
            Object o = application.getByUID("TestUID");
            EntityType type = (EntityType) allUids.get("TestUID");
            //Пример:
            if (type.equals(EntityType.DATE)){
                Date applicationDate = (Date) o;
                System.out.println(applicationDate.toString());
            }

            //Данные заявки

```

```

        JSONObject data = new
JSONObject(application.getAdditionalFields());

        //Добавление данных в сущность заявки если нужно добавить
какие-то значения
        data.put("PARAMS_HANDLER_MULTIPLE", "TestData");

        application.setAdditionalFields(data.toString());
        application.persist();

        //Сущность модели бизнес процесса
        BusinessModel businessModel =
_modelService.fetchBusinessModel(application.getBusinessModelId());

        //Сохраняем данные в инстанс камунды
        Map<String, Object> savedVars = new HashMap<>();
        savedVars.put("PARAMS_HANDLER_MULTIPLE", "213");

        //Завершить задачу с сохранением переменных
        externalTaskService.complete(externalTask, savedVars);

        //Если в системной форме прописана отправка сообщений по
событиям, то она реализуется здесь
        //Например, вот так отправляются сообщения о смене статуса
пользователям, задействованным в заявке
        if (application != null) {
            String statusId = application.getStatus();
            if (statusId != null && !statusId.isBlank()) {
                StatusModel status =
StatusModelLocalServiceUtil.fetchStatusModel(Long.parseLong(statusId));
                if (status.getSendMessages()) {
                    try {
                        List<Long> users =
UserParticipantLocalServiceUtil.findUsersByAppId(application.getId());
                        CamundaTaskUtil.sendStatusMessages(application, users, status, false);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }

    }).open();
}
@Deactivate
public void deactivate(){
    log.info("ExternalSimpleTask deactivated");
    client.stop();
}

```

```
    }  
}
```

### 3.2. Реализация пользовательских действий через расширение `AbstractServiceAction`

Расширение базового класса `AbstractServiceAction` позволяет использовать настраиваемые данные так же, как и в стандартных действиях. Такие действия необходимо регистрировать вместе со списком обычных действий.

При подписке требуется указать полное имя класса, например:

```
ru.internal.actions.external.TestServiceTask
```

Необходимо переопределить метод `getLabel`, чтобы задать отображаемое название действия:

```
@Override  
public String getLabel() {  
    return "Test service task";  
}
```

Также требуется переопределить метод `execute`, который отвечает за выполнение действия:

```
@Override  
public void execute(Application application, ExternalTask externalTask,  
CustomAction customAction) {  
    Map<String, Object> map = getSettings().asMap();  
    completeTask();  
}
```

В контексте доступны следующие методы:

- `completeTask()` – завершение системной формы;
- `completeTask(variables)` – завершение системной формы со списком параметров;
- `getSettings().asMap()` – получение настроек действия.

Пример реализации:

```
public class TestServiceTask extends AbstractServiceAction {  
  
    public TestServiceTask() {  
        super();  
        setSettingsSchema(loadSettingsDefinition());  
        setDefaultSettings(loadDefaultSettings());  
    }  
  
    @Override  
    public String getLabel() {  
        return "Test service task";  
    }  
}
```

```

@Override
public void execute(Application application, ExternalTask externalTask,
    CustomAction customAction) {
    Map<String, Object> map = getSettings().asMap();
    completeTask();
}

protected ActionSettingsSchema loadSettingsDefinition() {
    var schemaBuilder = new ActionSettingsSchema.Builder();
    schemaBuilder
        .textField()
        .label("Наименование организации")
        .key("orgNameSettingKey")
        .required()
        .build()
        .checkbox()
        .label("Включить пользователей в организацию")
        .key("includeUsersFlagKey")
        .right()
        .build()
        .textField()
        .label("123123")
        .key("testKey")
        .conditional("this.settings['orgNameSettingKey'] === 'xxx'")
        .right()
        .build();
    return schemaBuilder.build();
}

protected ActionSettings loadDefaultSettings() {
    return ActionSettings.of(
        Map.of(
            "includeUsersFlagKey", true,
            "orgNameSettingKey", "test"));
}
}

```

## 4. Реализация портлетов

Платформа позволяет размещать на страницах отдельные портлеты. Портлет может быть стандартным и входить в базовую функциональность платформы. Также вы можете добавить кастомный портлет с необходимой вам логикой.

В данном разделе рассматривается создание maven проекта портлета, включающего в себя web-модуль и service-модуль для взаимодействия с сущностями БД.

Сначала необходимо создать новый maven проект по аналогии с первым maven-приложением. В качестве archetype добавляем новый archetype, согласно следующим параметрам на рисунке 5.

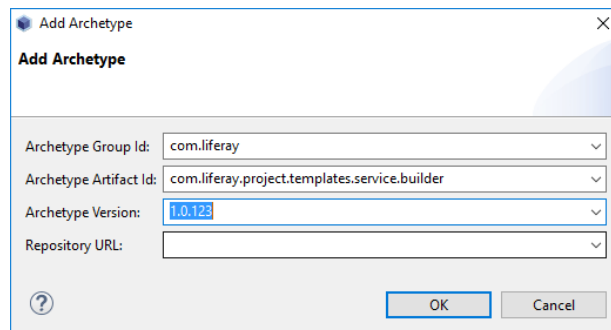


Рисунок 5 – Параметры шаблона проекта

После добавления archetype появится в общем списке (см. рис. 6).

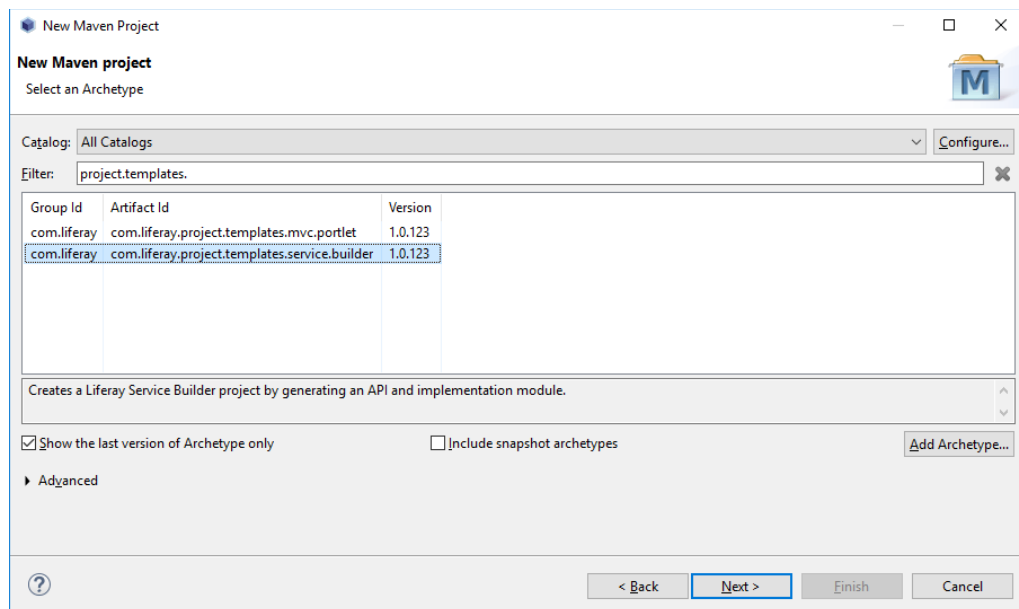


Рисунок 6 – Создание проекта на основе шаблона

Настраиваем service-builder портлет, согласно настройкам ниже (groupid, artifactid и version используем свои) на рисунке 7.

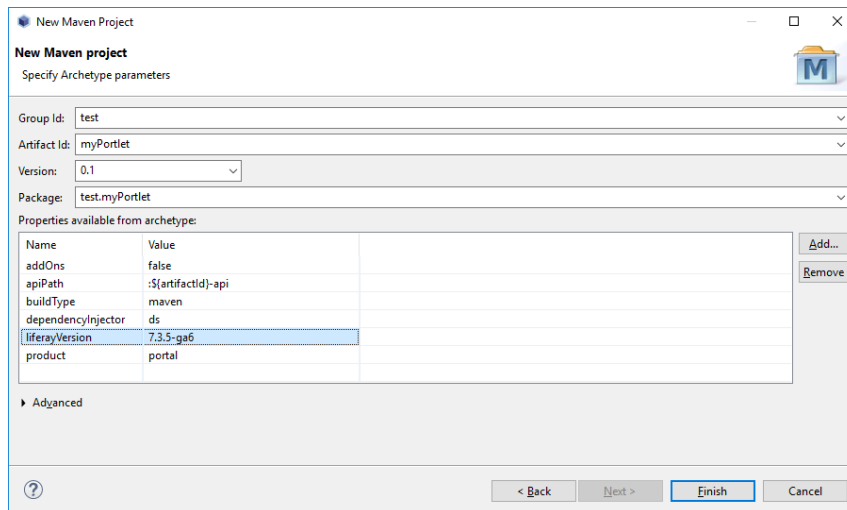


Рисунок 7 – Настройки service-builder портлета

В итоге, получается структура, изображенная на рисунке 8. Отметим, что service-builder template создает общую директорию, в которой хранится \*-api и \*-service составляющие портлета. Теперь нужно добавить \*-web составляющую (myportlet-web создает ранее по аналогии с первым приложением mvc-portlet).



Рисунок 8 – Структура проекта портлета

Закрываем в eclipse myportlet-web – в контекстном меню «Close project», и удаляем его из eclipse, не удаляя из системы (содержимое останется в системе, см. рис. 9).

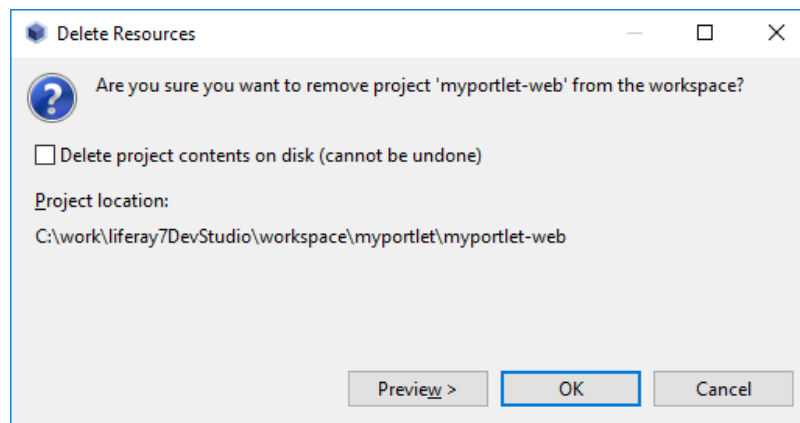


Рисунок 9 – Удаление myportlet-web

Перемещаем проект в myportlet, структура отображена на рисунке 10.

myportlet-api	04.05.2021 12:14	Папка с файлами	
myportlet-service	04.05.2021 12:15	Папка с файлами	
myportlet-web	04.05.2021 12:14	Папка с файлами	
.gitignore	04.05.2021 12:04	Текстовый докум...	1 КБ
.project	04.05.2021 12:04	Файл "PROJECT"	1 КБ
pom.xml	04.05.2021 12:16	Документ XML	3 КБ

Рисунок 10 – Структура проекта портлета

Далее импортируем maven-проект обратно в eclipse: file->import (см. рис. 11).

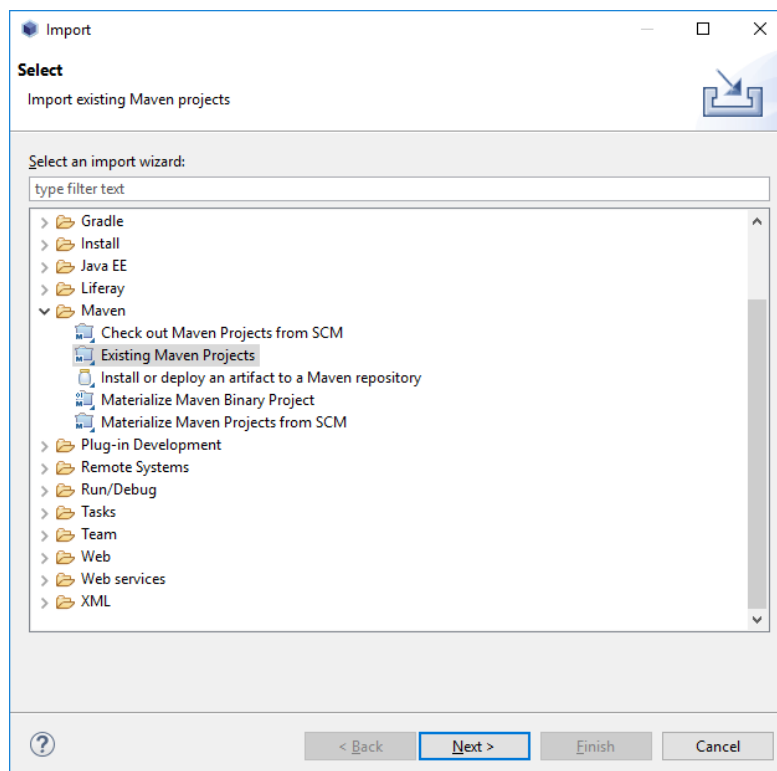


Рисунок 11 – Импорт maven проекта

Далее выбираем директорию, где находится созданный портлет (в нашем случае myportlet) и добавляем web-составляющую (см. рис. 12).

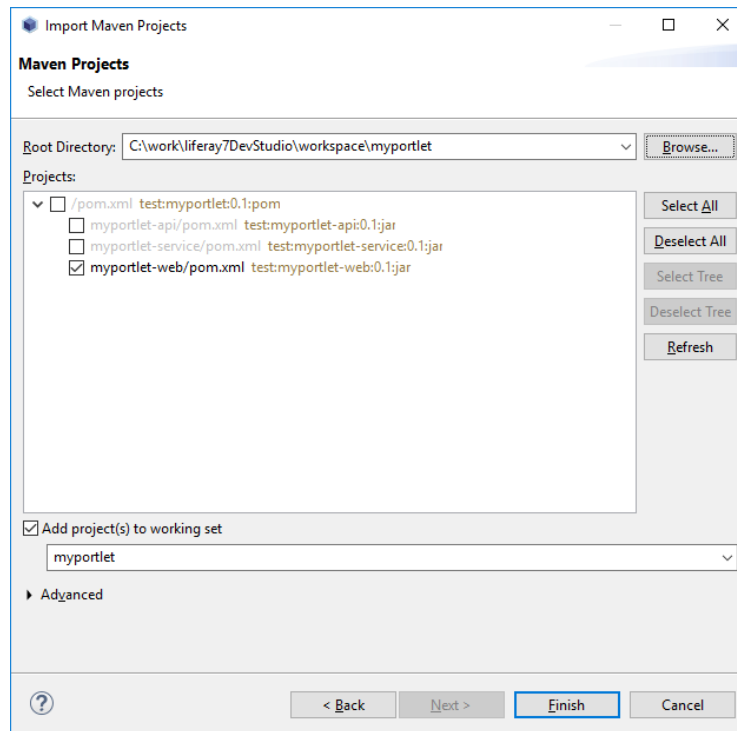


Рисунок 12 – Импорт maven проекта

Редактируем в портлет-web pom.xml, добавляем parent и добавляем api- составляющую в качестве зависимости (см. рис. 13).

```

<groupId>test</groupId>
<artifactId>myportlet-web</artifactId>
<version>0.1</version>
<parent>
  <groupId>test</groupId>
  <artifactId>myportlet</artifactId>
  <version>0.1</version>
</parent>
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
<dependencies>
  <dependency>
    <groupId>test</groupId>
    <artifactId>myportlet-api</artifactId>
    <version>0.1</version>
    <scope>provided</scope>
  </dependency>
</dependencies>

```

Рисунок 13 – Зависимости проекта

Редактируем содержимое myportlet – добавляем в качестве зависимости api для работы с классами портала и добавляем web-составляющую в качестве модуля (см. рис.14). Также убираем зависимость release.portal.api из pom модулей (\*-api, \*-service, \*-web).

```

-----
<groupId>test</groupId>
<artifactId>myportlet</artifactId>
<version>0.1</version>
<packaging>pom</packaging>
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
<dependencies>
  <dependency>
    <groupId>com.liferay.portal</groupId>
    <artifactId>release.portal.api</artifactId>
    <version>7.3.5-ga6</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
<modules>
  <module>myportlet-api</module>
  <module>myportlet-service</module>
  <module>myportlet-web</module>
</modules>
-----

```

Рисунок 14 – Изменение зависимостей

Для проверки корректности сборки общего проекта, запускаем build-service у \*-service (см. рис. 15).

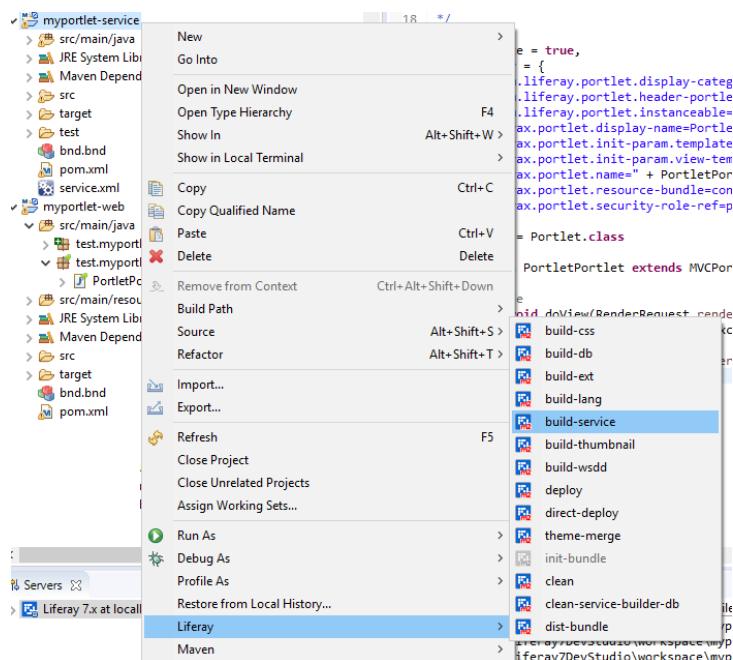


Рисунок 15 – Запуск build-service

Если все выполнено успешно, в консоли появится соответствующее сообщение (см. рис. 16).

```

Writing C:\work\liferay7DevStudio\workspace\myportlet\myportlet-service\src\main\java\test\myportlet\service\persistence\impl\c
Writing C:\work\liferay7DevStudio\workspace\myportlet\myportlet-service\src\main\resources\META-INF\sql\indexes.sql
Writing C:\work\liferay7DevStudio\workspace\myportlet\myportlet-service\src\main\resources\META-INF\sql\tables.sql
Writing C:\work\liferay7DevStudio\workspace\myportlet\myportlet-service\src\main\resources\service.properties
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.266 s
[INFO] Finished at: 2021-05-04T12:22:21+07:00
[INFO] -----

```

Рисунок 16 – Сообщение об успешном выполнении в консоли

По умолчанию в service.xml хранится описание сущности Foo, попробуем вызвать соответствующую сущность из модуля \*-web:

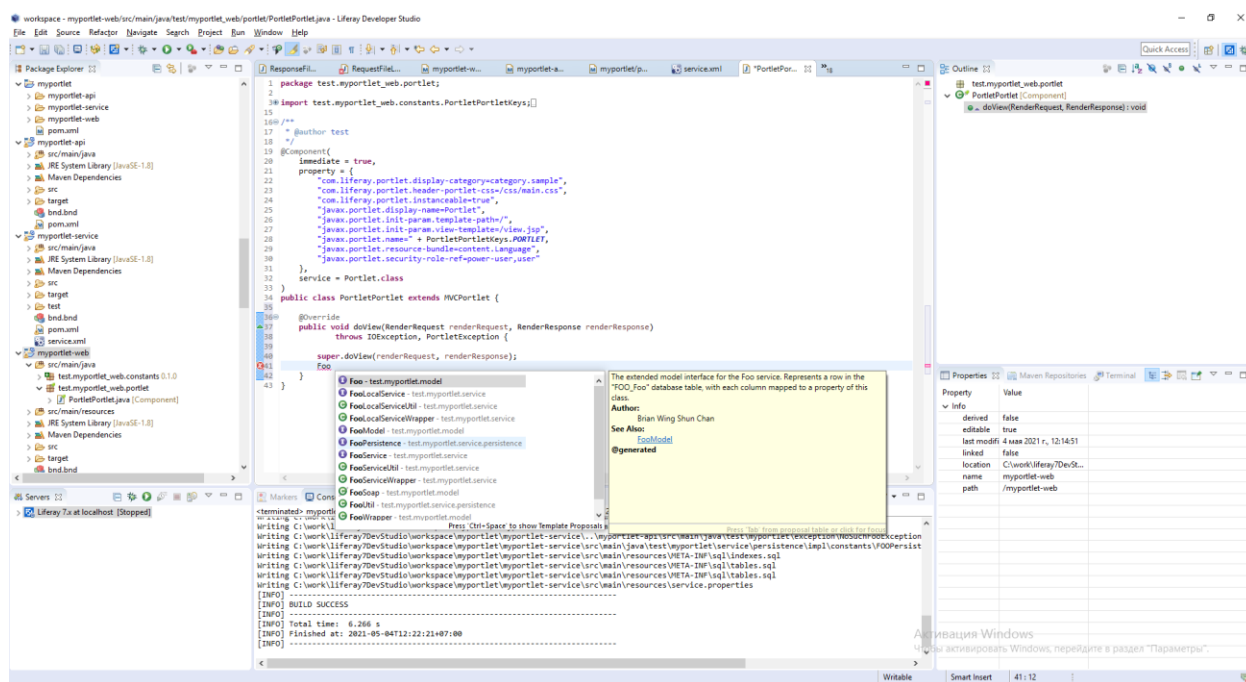


Рисунок 17 – Обращение сущности Foo из модуля