

Общество с ограниченной ответственностью «Си Икс Лаб»

РЕШЕНИЕ ПО СОЗДАНИЮ АВТОМАТИЗИРОВАННЫХ РАБОЧИХ МЕСТ

СХВОХ V6.0

ОПИСАНИЕ ФУНКЦИОНАЛЬНЫХ ХАРАКТЕРИСТИК

Листов 40

Москва

2024

В настоящем документе приведено описание научно-технического продукта «Решение по созданию автоматизированных рабочих мест (СХВОХ) v6.0» (далее – Система).

СОДЕРЖАНИЕ

ТЕРМИНЫ И СОКРАЩЕНИЯ	4
1. ОБЩИЕ СВЕДЕНИЯ	6
1.1. НАИМЕНОВАНИЕ И УСЛОВНОЕ ОБОЗНАЧЕНИЕ СИСТЕМЫ.....	6
1.2. ЯЗЫКИ ПРОГРАММИРОВАНИЯ, НА КОТОРЫХ НАПИСАНА СИСТЕМА.....	6
2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ	7
2.1. Ядро	8
2.2. ДОПОЛНЕНИЯ РАЗРАБОТЧИКА	10
2.3. СИСТЕМНЫЕ ДОПОЛНЕНИЯ	11
2.4. БИЗНЕС-ДОПОЛНЕНИЯ	11
2.5. ДОПОЛНЕНИЯ АДМИНИСТРАТОРА	12
3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ.....	13
3.1. АЛГОРИТМ ПРОГРАММЫ	13
3.1.1. Ядро.....	13
3.1.2. Дополнения разработчика	17
3.1.3. Системные дополнения	33
3.1.4. Бизнес-дополнения	36
3.1.5. Дополнение администратора.....	39

ТЕРМИНЫ И СОКРАЩЕНИЯ

Термин/сокращение	Определение/расшифровка
DTO	Data Transfer Object – шаблон проектирования, используется для передачи данных между подсистемами приложения
JSON	JavaScript Object Notation – текстовый формат обмена данными, основанный на JavaScript
S3	Simple Storage Service – веб-служба, предоставляющая возможность хранения и получения любого объема данных в любое время из любой точки сети
ЭВМ	Электронно-вычислительная машина
Приложение	Клиентское приложение использующее CXBOX v6.0
API	Application Programming Interface Описание способов взаимодействия одной компьютерной программы с другими
CRUD	Create, Read, Update, Delete – создание, чтение, модификация, удаление
hotkey	«Горячая» клавиша, комбинация клавиш на клавиатуре, нажатие на которые позволяет выполнять различные действия
HTTP	HyperText Transfer Protocol Протокол прикладного уровня передачи данных
java IDE	GigaIDE / IntelliJ IDEA
JDBC	Java DataBase Connectivity Соединение с базами данных на Java
quick-fix	Исправления, доступные в один клик
REST API	API, который использует HTTP-запросы для получения, извлечения, размещения и удаления данных

Термин/сокращение	Определение/расшифровка
SMTP	Simple Mail Transfer Protocol Простой протокол передачи почты
SSO	Single Sign-On Технология единого входа — технология, при использовании которой пользователь переходит из одного раздела портала в другой, либо из одной системы в другую, не связанную с первой системой, без повторной аутентификации
STARTTLS	Расширение протокола текстового обмена, позволяющее создать зашифрованное соединение поверх обычного TCP-соединения
Tomcat	Контейнер сервлетов с открытым исходным кодом
UI	User Interface, пользовательский интерфейс
БК, bc	Бизнес-компонента
Виджет, widget	Элемент интерфейса, состоящий из полей, обычно содержит набор данных для одной операции
Вью, view	Элемент интерфейса, включающий в себя виджеты, поля, меню навигации.
Экран, screen	Элемент интерфейса, содержащий несколько вью
Поле	Элемент интерфейса, содержащий информацию по одному атрибуту бизнес сущности
ПО	Программное обеспечение
Система, Решение	Программа для ЭВМ «Решение по созданию автоматизированных рабочих мест (СХВОХ) v 6.0»
СУБД	Система управления базами данных

1. Общие сведения

1.1. Наименование и условное обозначение системы

Полное наименование: программа для ЭВМ «Решение по созданию автоматизированных рабочих мест v 6.0».

Краткое наименование: CXBOX v6.0.

1.2. Языки программирования, на которых написана система

CXBOX v6.0 разработан с использованием современных инструментов для создания масштабируемых Enterprise-приложений. Backend часть реализована с использованием Java и фреймворка Spring, обеспечивающего надежность, безопасность и гибкость системы.

Java, широко используемый язык программирования, обеспечивает высокую производительность и позволяет создавать надежные и эффективные решения. В сочетании с фреймворком Spring, который предоставляет широкий спектр инструментов и функций, разработчики смогли создать мощный и гибкий backend. Spring обеспечивает удобное управление зависимостями, обработку HTTP-запросов, работу с базами данных, безопасность и другие важные функции, необходимые для создания качественного веб-приложения.

Frontend часть разработана с использованием TypeScript и фреймворка ReactJS. TypeScript, статически типизированный язык программирования, добавляет дополнительную степень безопасности и инструменты для разработки на JavaScript. Это позволяет создавать структурированный, надежный и масштабируемый код.

ReactJS, один из самых популярных фреймворков для разработки пользовательских интерфейсов, обеспечивает высокую производительность, переиспользуемость компонентов и удобную работу с состоянием приложения. Благодаря своей эффективной виртуальной DOM, ReactJS обеспечивает быстрое обновление пользовательского интерфейса и плавные интерактивные возможности.

В итоге, комбинация Java и Spring на бэкенде, а также TypeScript и ReactJS на фронтенде, создает мощную и современную архитектуру для создания любых Enterprise-приложений, способствующую эффективной разработке, масштабированию и поддержке проектов на базе CXBOX v6.0.

2. Функциональное назначение

Решение CXBOX v6.0 представляет из себя систему для декларативного создания интерфейсов, автоматизации рабочих мест и интеграции в ландшафт заказчика с использованием готовых функциональных блоков. Под “декларативным” созданием интерфейсов понимается подход, при котором элементы интерфейса автоматизированного рабочего места описываются в соответствии с предусмотренным системой форматом, а система отвечает за интерпретацию полученного описания, отрисовку интерфейса автоматизированного рабочего места и реализацию логики работы элементов интерфейса.

Создание приложения на базе CXBOX v6.0 имеет следующие преимущества:

- экономия ресурсов разработки в проектах, включающих задачи разработки интерфейсов автоматизированных рабочих мест предприятий;
- возможность быстрой автоматизации рабочих мест за счет использования готовых функциональных блоков;
- возможность быстрой интеграции приложений на базе CXBOX v6.0 с другими системами клиентов;
- возможность автоматической проверки кода и исправления ошибок, а также использования интерактивной документации;
- решение проблемы дефицита квалифицированных программистов вследствие использования наиболее популярного и современного стека технологий, полностью поддерживающего собственную разработку;
- совместимость с новейшими подходами в собственной разработке, включая agile, ci/cd, микросервисную архитектуру;
- открытость, то есть возможность расширять решение собственными функциональными дополнениями для последующего переиспользования;
- возможность простой адаптации предложенных в решении компонент в соответствии со специфическими потребностями предприятия.

Решение состоит из следующих архитектурных частей:

- ядро, обеспечивающее реализацию интерфейса приложения и расширяемое дополнениями;
- бизнес-дополнения, интегрируемые с ядром бэкенда и реализующие бизнес-логику приложения;
- дополнения разработчика, интегрируемые со средой разработки java IDE и предназначенные для ускорения и упрощения процесса разработки;
- административные дополнения, реализующие задачи администратора приложения;

- системные дополнения, реализующие технические и интеграционные решения.

2.1. Ядро

Ядро состоит из элементов: поле, виджет, экран, вью, меню навигации, – на основе которых осуществляется построение интерфейса.

Основные возможности ядра решения:

- создание неограниченного количества экранов приложения;
- на каждом экране возможно добавление нескольких представлений (view);
- каждое представление может содержать неограниченное количество виджетов;
- виджеты на представлении могут быть как независимыми, так и связанными; связь виджетов образует иерархию «родитель-ребенок», количество уровней (глубина иерархии) не ограничено;
- типы виджетов представлены формами, таблицами/списками, всплывающими информационными виджетами, всплывающими списками единичного и множественного выбора, но не ограничены ими; новые типы виджетов могут быть добавлены без доработки ядра на уровне каждого проекта;
- виджет может содержать неограниченное количество полей;
- доступные типы полей представлены текстовыми полями, числовыми полями, полями для ввода даты, чекбоксами, справочниками, радиокнопками, полями выбора связанной сущности, полями для работы с файлами, но не ограничены ими; новые типы поле могут быть добавлены без доработки ядра на уровне каждого проекта;
- поля виджета могут быть связанными, то есть при изменении значения одного из полей могут изменяться значения в другом поле или изменяться набор доступных значений в другом поле;
- логика связи полей описывается на бэкенде без кастомизации фронтенда;
- поля могут быть как редактируемыми, так и не редактируемыми. Редактируемость полей настраивается на бэкенде без кастомизации фронтенда. Редактируемость полей может быть контекстно-зависимой (например, в зависимости от статуса задачи, типа задачи, роли пользователя и т.д.);
- поля могут быть как обязательными, так и необязательными. Обязательность полей настраивается на бэкенде без кастомизации фронтенда. Обязательность полей может быть контекстно-зависимой (например, в зависимости от статуса задачи, типа задачи, роли пользователя и т.д.);
- на табличных виджетах поддерживается возможность фильтрации по любому полю;

- на табличных виджетах поддерживается кастомизация фильтрации по любому полю на стороне бэкенда без необходимости кастомизации фронтенда.
- на табличных виджетах поддерживается возможность сортировки по любому полю;
- на табличных виджетах поддерживается пагинация (постраничный вывод данных) с возможностью перейти на нужную страницу. С бэкенда данные запрашиваются и приходят по одной странице;
- на табличных виджетах поддерживается возможность использования предустановленных фильтров;
- взаимодействие фронтенда и бэкенда происходит по универсальному контракту. Набор методов, порядок их вызова, структура запросов и ответов едины для всех типов виджетов и не требуют проектирования и создания с нуля для каждого экрана/представления/виджета;
- поддерживается создание всех методов универсального контракта REST API для виджета в одну строку без необходимости создания стандартного spring boot rest-контроллера;
- поддерживается администрирование справочников через интерфейс без доработки на бэкенде или фронтенде;
- на виджете может быть неограниченное количество действий (кнопок). Их добавление выполняется на бэкенде без кастомизации фронтенда;
- кнопки, отображаемые на виджете, могут быть как контекстно-независимыми, так и зависящими от текущей записи/авторизованного пользователя (например, доступные действия могут зависеть от типа задачи, статуса задачи, роли пользователя и т.д.);
- поддерживается несколько типов валидации: простая валидация на заполненность обязательных полей при сохранении записи; валидация произвольной сложности, настраиваемая на бэкенде и вызываемая по кнопке или при сохранении записи, с наличием возможности отображения сообщения для пользователя;
- взаимодействие фронтенда и бэкенда выполняется через REST API посредством стандартных запросов http (get, post, put, delete) с использованием json для простоты отладки и доработки;
- управление составом полей выполняется без участия фронтенд-разработчика;
- контракт стандартизирован, бэкенд-разработчик имеет возможность описывать внешний вид и состав полей с помощью json-спецификации;

- используется стандартная структура spring boot-проекта (структура пакетов, именованые классы, использование библиотек) с целью понижения порога входа java-разработчиков с опытом spring boot;
- предусмотрены точки расширения для подключения альтернативных библиотек визуальных элементов за счет разделения библиотеки фронт-енда на условно смысловые блоки: логическое поведение и визуальное представление;
- предоставляет возможность получения, создания, изменения и удаления данных не только через загрузку данных в собственную DB (базу данных), но и посредством API для взаимодействия с внешними источниками данных.

2.2. Дополнения разработчика

Дополнения разработчика состоят из: шаблонизатора кода, дополнения быстрой навигацией, интерактивной документацией, навигацией к интерактивной документации, автодополнения кода, поиска и исправления ошибок, инструмента мониторинга, что позволяет ускорять и упрощать процесс разработки

Основные возможности дополнений разработчика:

- интеграция в наиболее широко используемую среду разработки Java – IntelliJ IDEA;
- предоставление возможности перегрузки (hot reload) интерфейса при изменении метаданных в IDE без необходимости перезапуска проекта;
- предоставление по hotkey шаблонов создания типовых элементов UI;
- предоставление для выбора в визуальном интерфейсе разработчика шаблонов создания типовых элементов UI;
- проверка синтаксиса, консистентности, автозаполнения, быстрой навигации для кода метаданных для заведения экранов, представлений, виджетов и полей;
- предоставление расширенного набора инспекций (проверки типовых ошибок разработчика);
- предоставление автоматических quick-fix (исправления, доступные в один клик) для выявляемых инспекциями ошибок;
- предоставление возможности просмотра интерфейса в режиме отладки (debug mode) для быстрого перехода из элементов интерфейса к коду, в котором они декларируются;
- возможность мониторинга работоспособности приложения;

- интерактивная документация по работе с элементами Системы;
- переход из кода проекта к интерактивной документации с описанием элементов Системы;
- помощь в написании кода и метаданных с помощью предложения ключевых слов или объектов в зависимости от контекста.

2.3. Системные дополнения

Системные дополнения состоят из: «S3 File Storage», интеграционного модуля, базовой авторизации, SSO-авторизации, поддержки различных СУБД.

Основные возможности системных дополнений:

- обеспечение возможности загружать файлы через интерфейсы в файловые хранилища;
- поддержка локальных хранилищ (на уровне файловой системы сервера приложений) и облачных S3-совместимых хранилищ;
- конфигурация приложения для хранения файлов в локальном хранилище или для использования инструментов подключения к S3-хранилищу выбранного провайдера;
- одновременная работа с несколькими хранилищами с возможностью программного управления выбором хранилища;
- готовый компонент для сохранения и получения файла из хранилища;
- готовый компонент для загрузки файла на устройство пользователя;
- поле, готовое для применения на виджетах приложения и реализующее процесс выбора пользователем файла и сохранения его в хранилище, с различными параметрами настройки и управления процессом сохранения;
- функция авторизации пользователя в приложении на основании данных из сторонней системы (провайдера) аутентификации;
- предоставление универсального контракта, позволяющего внешним системам взаимодействовать с сущностями модели данных и бизнес-логикой без написания собственных REST-контроллеров на BE-части.

2.4. Бизнес-дополнения

Бизнес-дополнения состоят из: интеграции с email, push-уведомлений, предварительного просмотра документов.

Основные возможности бизнес-дополнений:

- предоставление готовых дополнений с профессиональной поддержкой, позволяющих реализовать типовые бизнес-требования за считанные минуты, а не месяцы;

- возможность отправки уведомления с BE-части на FE-часть о работе асинхронных операций;
- возможность интеграции с системами электронной почты для направления пользователям email-сообщений;
- возможность предварительного просмотра документов/изображений в приложении без необходимости их скачивания.

2.5. Дополнения администратора

Дополнение администратора состоит из планировщика заданий, что позволяет запускать процессы приложения по расписанию.

Основные возможности дополнений администратора:

- выполнение обслуживания базы данных по установленному расписанию;
- выполнение анализа наполненных данных по расписанию и отправка уведомлений по результатам анализа данных, а также составление отчетов по накопленным данным;
- рассылка email-сообщений в указанное время и дату;
- позволяет задавать временные интервалы, периодичность и условия выполнения задач;
- позволяет выполнять сложные цепочки заданий за счет установки последовательности выполнения задач и установки условия для их запуска;
- позволяет отслеживать состояние выполнения заданий, фиксировать ошибки и предупреждения, а также вести журнал выполнения заданий, что облегчает диагностику и устранение неисправностей.

3. Описание логической структуры

3.1. Алгоритм программы

3.1.1. Ядро

Система должна поддерживать на экранных формах поля следующих типов:

- текст;
- многострочное текстовое поле;
- число;
- дата;
- дата и время;
- чекбокс;
- справочник;
- выбор объекта;
- радиокнопка;
- файл.

Текстовое поле (рисунок 1) позволяет ввести произвольное текстовое значение.

Address



approximate address

Рисунок 1

Многострочное текстовое поле (Рисунок 2) позволяет ввести большое произвольное текстовое значение, включая несколько строк и абзацев.

Address



approximate address

Рисунок 2

При нажатии в правый нижний угол возможно увеличить размер поля.

Числовое поле (рисунок 3) позволяет ввести только числовое значение.

Sum

1111|

Рисунок 3

Поле даты (рисунок 4) позволяет ввести дату.

Дата

12.04.2023



Рисунок 4

При нажатии на поле отображается календарь (рисунок 5).

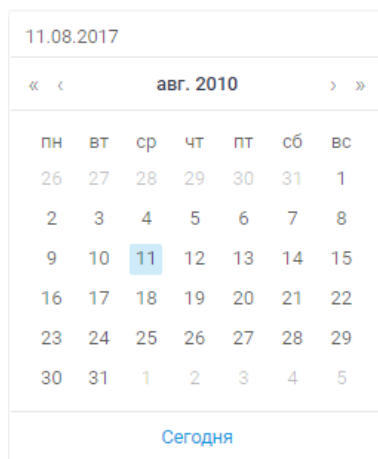


Рисунок 5

Пользователь может выбрать дату двумя способами:

- ввести вручную нужное значение в поле календаря;
- найти дату в календаре.

В календаре должны быть предусмотрены возможности:

- перехода к предыдущему и следующему месяцу;
- перехода к предыдущему и следующему году;
- открытия полного списка месяцев года для быстрого выбора;
- открытия полного списка лет текущего десятилетия для быстрого выбора;
- перехода к предыдущему и следующему десятилетию;
- выбора текущей даты.

Поле даты/времени (рисунок 6) позволяет ввести дату и время с точностью до минуты.

Start Date


 

Рисунок 6

При нажатии на поле отображается календарь с кнопкой «Выбрать время» (рисунок 7).

01.08.2024 10:02

<< < Aug 2024 > >>

Su	Mo	Tu	We	Th	Fr	Sa
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Now select time **Ok**

Рисунок 7

В календаре с выбором времени должны поддерживаться функции, аналогичные календарю выбора даты, а также возможность открытия списков часов и минут (рисунок 8) по кнопке «Выбрать время».

01.08.2024 10:02

Aug 1 2024

10	02
11	03
12	04
13	05
14	06
15	07
16	08
17	09
18	10
19	11

Now select date **Ok**

Рисунок 8

В календаре с выбором времени должна быть предусмотрена возможность выбора текущего времени (с точностью до минуты).

Чекбокс (рисунок 9) представляет собой логический флаг, который может иметь два состояния: включен и выключен.



Рисунок 9

Поле с типом справочника (рисунок 10) позволяет выбрать одно из фиксированных значений из выпадающего списка.

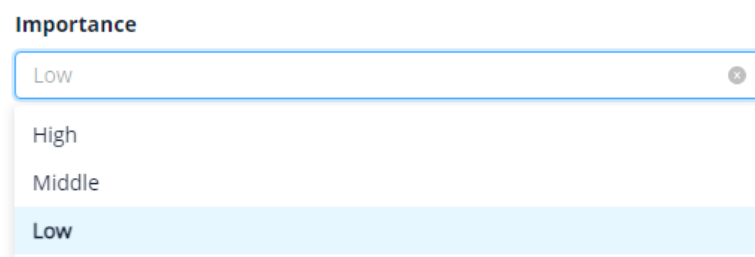


Рисунок 10

При ручном вводе текста в поле в списке должна производиться фильтрация значений: должны отображаться только значения, содержащие введенный текст.

Поле с выбором объекта (рисунок 11) позволяет отобразить во всплывающем окне список объектов и выбрать один из них.

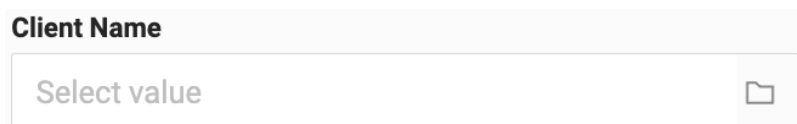


Рисунок 11

Радиокнопка (рисунок 12) позволяет выбрать одно из predetermined значений, отображаемых в виде переключателей.

Radio

- First value
- Second value
- Third value
- Fourth value

Рисунок 12

При переключении значений, ранее выбранное значение должно сбрасываться.

Поле файл (Рисунок 13) позволяет прикреплять и просматривать один файл.



Рисунок 13

При нажатии на «три точки» должно открываться диалоговое окно локальной файловой системы для выбора загружаемого файла.

При нажатии на загруженный файл в поле, должно происходить скачивание файла.

3.1.2. Дополнения разработчика

3.1.2.1. Шалонизатор и автодополнение кода

3.1.2.1.1. Добавление представления на экран

Предусловие:

- создан JSON-файл с описанием экрана.

Основной сценарий:

1. Пользователь открывает JSON-файл с описанием экрана.
2. Система отображает иконку добавления в строках свойства "menu" и свойства "child" каждого агрегата представлений.
3. Пользователь нажимает на иконку добавления.
4. Система открывает меню выбора действия: одиночное представление, агрегат представлений.
5. Пользователь выбирает пункт добавления одиночного представления.
6. Система добавляет в свойство перечня представлений экрана новый объект представления.
7. Система открывает выпадающий список имен созданных представлений в проекте.
8. Пользователь вводит имя нового представления.
9. Система выделяет имя представления как несуществующего.
10. Система отображает иконку функции коррекции.
11. Пользователь нажимает на иконку функции коррекции.
12. Система открывает меню выбора действия с возможностью создания файла описания представления.
13. Пользователь выбирает пункт создания файла описания представления.

14. Система создает JSON-файл с именем <имя представления>.view.json, содержащий описание представления.

3.1.2.1.2. Добавление агрегата представлений на экран

Предусловие:

- создан JSON-файл с описанием экрана.

Основной сценарий:

1. Пользователь открывает JSON-файл с описанием экрана.
2. Система отображает иконку добавления в строках свойства "menu" и свойства "child" каждого агрегата представлений.
3. Пользователь нажимает на иконку добавления.
4. Система открывает меню выбора действия: одиночное представление, агрегат представлений.
5. Пользователь выбирает пункт добавления агрегата представлений.
6. Система добавляет в свойство перечня представлений экрана новый объект с заголовком и внутренним перечнем представлений.

3.1.2.1.3. Добавление виджета на представление

Предусловие:

- создан JSON-файл с описанием представления.

Основной сценарий:

1. Пользователь открывает JSON-файл с описанием представления.
2. Система отображает иконку добавления в строке ключа свойства перечня виджетов.
3. Пользователь нажимает на иконку добавления.
4. Система добавляет в свойство перечня виджетов представления новый объект виджета.
5. Система открывает выпадающий список имен созданных виджетов в проекте.
6. Пользователь вводит имя нового виджета.
7. Система выделяет имя виджета как несуществующего.
8. Система отображает иконку функции коррекции.
9. Пользователь нажимает на иконку функции коррекции.
10. Система открывает меню выбора действия с возможностью создания файла описания виджета.
11. Пользователь выбирает пункт создания файла описания виджета.

12. Система создает JSON-файл с именем <имя виджета>.widget.json, содержащий описание виджета.

3.1.2.1.4. Добавление поля на виджет

Предусловие:

- создан JSON-файл с описанием виджета.

Основной сценарий:

1. Пользователь открывает JSON-файл с описанием виджета.
2. Система отображает иконку добавления в строке ключа свойства перечня полей.
3. Пользователь нажимает на иконку добавления.
4. Система отображает меню типов полей.
5. Пользователь выбирает тип поля.
6. Система добавляет в свойство перечня полей виджета новый объект поля.
7. Пользователь вводит имя нового поля.
8. Система выделяет имя поля как несуществующего.
9. Система отображает иконку функции коррекции.
10. Пользователь нажимает на иконку функции коррекции.
11. Система открывает меню выбора действия с возможностью добавления кода для поля.
12. Пользователь выбирает пункт добавления кода для поля.
13. Система добавляет необходимый код для работы поля во всех слоях приложения:
 - поле DTO;
 - поле Entity;
 - отображение данных от Entity к DTO (поддерживается отображение конструктора DTO и отображение класса Java Bean);
 - отображение данных от DTO к Entity в сервисном слое;
 - код в мета-сервисе, который делает поле редактируемым и фильтруемым.

3.1.2.1.5. Автозаполнение ключа поля из списка полей сущности

Предусловие:

- создан файл DTO с описанием полей;
- создан JSON-файл с описанием виджета (widget.json).

Основной сценарий:

1. Пользователь открывает файл widget.json.

2. Пользователь начинает вводить значение в массиве `fields` в ключе `key` или нажимает `Ctrl + Пробел` до ввода значения.
3. Система показывает выпадающий список доступных ключей, объявленных в DTO.
4. Пользователь выбирает значение из списка, и поле с ключом `key` заполняется в соответствии со значением из файла DTO.

3.1.2.1.6. Автозаполнение параметров свойства условий отображения виджета

Предусловие:

- создан файл DTO с описанием полей;
- создан JSON-файл с описанием виджета (`widget.json`);
- в файле `widget.json` в массиве `fields` объявлены поля, соответствующие описанным в DTO.

Основной сценарий:

1. Пользователь открывает файл `widget.json`.
2. Пользователь переходит к разделу `layout`, определяющий порядок отображения полей, перечисленных в массиве `fields`.
3. Пользователь начинает вводить значение в массиве `layout` в ключе `fieldKey` или нажимает `Ctrl + Пробел` до ввода значения.
4. Система отображает выпадающий список возможных ключей из массива `fields`.
5. Пользователь выбирает подходящий ключ из списка.
6. Система вставляет выбранное значение.

3.1.2.1.7. Автозаполнение в списке операций

Предусловие:

- создан JSON-файл с описанием виджета (`widget.json`);
- создан Java-класс с описанием операций для соответствующей бизнес-компоненты (bc).

Основной сценарий:

1. Пользователь открывает файл `widget.json`.
2. Пользователь переходит к разделу `options -> actionGroups -> include`, где указываются действия, отображаемые на виджете.
3. Пользователь начинает вводить название действия в массиве `include` или нажимает `Ctrl + Пробел` до ввода значения.
4. Система отображает выпадающий список возможных действий, определенных в Java-классе для текущей бизнес-компоненты.
5. Пользователь выбирает подходящее действие из списка.

6. Системы вставляет выбранное действие.

3.1.2.1.8. Автозаполнение имени бизнес-компоненты списка значений для выбора в описании виджета

Предусловия:

- открыт JSON-файл описания виджета;
- виджет содержит в одном из полей вызов списка значений для выбора;
- в строке свойства "popupVcName" поля списка значений доступна иконка дополнительных функций.

Основной сценарий:

1. Пользователь нажимает на иконку дополнительных функций в строке "popupVcName".
2. Система открывает выпадающий список с именами бизнес-компонент.
3. Пользователь выбирает имя бизнес-компоненты.
4. Система подставляет имя бизнес-компоненты в значение свойства "popupVcName".

3.1.2.1.9. Автозаполнение левой части карты списка значений для выбора в описании виджета

Предусловия:

- открыт JSON-файл описания виджета;
- виджет содержит в одном из полей вызов списка значений для выбора;
- в строках полей свойства "pickMap" поля списка значений доступна иконка дополнительных функций.

Основной сценарий:

1. Пользователь выделяет левую часть поля свойства "pickMap" и нажимает на иконку дополнительных функций в строке.
2. Система открывает выпадающий список с возможными значениями.
3. Пользователь выбирает значение.
4. Система подставляет значение в левую часть поля свойства "pickMap".

3.1.2.1.10. Автозаполнение правой части карты списка значений для выбора в описании виджета

Предусловия:

- открыт JSON-файл описания виджета;
- виджет содержит в одном из полей вызов списка значений для выбора;

- в строках полей свойства "pickMap" поля списка значений доступна иконка дополнительных функций.

Основной сценарий:

1. Пользователь выделяет правую часть поля свойства "pickMap" и нажимает на иконку дополнительных функций в строке.
2. Система открывает выпадающий список с возможными значениями.
3. Пользователь выбирает значение.
4. Система подставляет значение в правую часть поля свойства "pickMap".

3.1.2.2. Дополнение быстрой навигации

3.1.2.2.1. Навигация из widget.json в Java-код бизнес-компоненты

Предусловие:

- создан JSON-файл с описанием виджета (widget.json)
- в описании виджета указана строка, содержащая имя бизнес-компоненты (bcName).

Основной сценарий:

1. Пользователь открывает файл widget.json.
2. Система отображает иконку дополнительных функций в строке, где указано имя бизнес-компоненты в ключе bcName.
3. Пользователь нажимает на иконку.
4. Система открывает файл Java-кода, где объявлена указанная бизнес-компонента.

3.1.2.2.2. Навигация из Java-кода бизнес-компоненты в связанные виджеты

Предусловие:

- создан Java-код с описанием бизнес-компоненты.

Основной сценарий:

1. Пользователь открывает Java-код с описанием бизнес-компоненты.
2. Пользователь наводит курсор на имя бизнес-компоненты и нажимает Ctrl + Click.
3. Система отображает список всех виджетов, которые используют эту бизнес-компоненту.
4. Пользователь выбирает виджет из списка.
5. Система открывает соответствующий JSON-файл с описанием виджета (widget.json).

3.1.2.2.3. Навигация из widget.json в DTO

Предусловие:

- создан JSON-файл с описанием виджета (widget.json);
- в описании виджета указан массив fields, в котором перечислены поля, определенные в соответствующем файле DTO;

Основной сценарий:

1. Пользователь открывает файл widget.json.
2. Система отображает иконку дополнительных функций в строке, где указан ключ виджета “key”.
3. Пользователь нажимает на иконку.
4. Система открывает соответствующий файл DTO, где объявлено указанное поле.

3.1.2.2.4. Навигация из DTO в widget.json

Предусловие:

- создан файл DTO с описанием полей;

Основной сценарий:

1. Пользователь открывает файл DTO с описанием полей.
2. Пользователь наводит курсор на объявленное поле и нажимает Ctrl + Click.
3. Система открывает JSON-файл с описанием виджета (widget.json), где это поле используется в массиве fields.

3.1.2.2.5. Навигация из widget.json в view.json

Предусловие:

- создан JSON-файл с описанием виджета (widget.json);
- в описании виджета указана строка с именем виджета в ключе name;

Основной сценарий:

1. Пользователь открывает файл widget.json.
2. Система отображает иконку дополнительных функций в строке, где указано название виджета в ключе name.
3. Пользователь нажимает на иконку.
4. Система открывает JSON-файл с описанием представления (view.json), где указано использование виджета.

3.1.2.2.6. Навигация из view.json в widget.json

Предусловие:

- создан JSON-файл с описанием представления (view.json);
- в описании представления указано имя виджета в ключе widgetName;

Основной сценарий:

1. Пользователь открывает файл view.json.
2. Система отображает иконку дополнительных функций в строке с указанием имени виджета в ключе widgetName в массиве widgets.
3. Пользователь нажимает на иконку.
4. Система открывает соответствующий файл JSON с описанием виджета (widget.json).

3.1.2.2.7. Навигация из view.json в screen.json

Предусловие:

- создан JSON-файл с описанием представления (view.json);
- в описании представления указано имя представления в ключе name;

Основной сценарий:

1. Пользователь открывает файл view.json.
2. Система отображает иконку дополнительных функций в строке с указанием имени представления в ключе name.
3. Пользователь нажимает на иконку.
4. Система открывает JSON-файл с описанием экрана (screen.json), где указано представление в составе экрана.

3.1.2.2.8. Навигация из screen.json в view.json

Предусловие:

- создан JSON-файл с описанием экрана (screen.json);
- в описании экрана указано представление (view) в составе экрана.

Основной сценарий:

1. Пользователь открывает файл screen.json.
2. Система отображает иконку дополнительных функций в строке, где указывается название представления в ключе viewName в массиве views.
3. Пользователь нажимает на иконку.
4. Система открывает JSON-файл с описанием представления (view.json).

3.1.2.3. Интерактивная документация и навигация к документации

3.1.2.3.1. Просмотр списка типов полей экранных форм в интерактивной документации

Предусловие:

- пользователь находится на сайте с документацией Решения.

Основной сценарий:

1. Пользователь открывает раздел документации, посвященный полям экранных форм.
2. Система отображает список всех типов полей, доступных для использования в приложении.
3. Пользователь выбирает интересующий тип поля для более детального изучения.

3.1.2.3.2. Работа с описанием типа поля в интерактивной документации

Предусловие:

- пользователь находится на сайте с документацией Решения;

- открыта страница с описанием конкретного типа поля.

Основной сценарий:

1. Пользователь открывает страницу с описанием выбранного типа поля.
2. Система отображает следующую информацию:
 - Наличие текста-заполнителя;
 - Цвет;
 - Запрет и разрешение редактирования;
 - Фильтрация;
 - Гиперссылка;
 - Контроль вводимых значений.
3. Пользователь выбирает свойство на странице описания типа поля.
4. Система отображает пример применения данного свойства:
 - Рисунок с отображением поля, где используется данное свойство;
 - Пример исходного кода для настройки свойства.

3.1.2.3.3. Переход из описания виджета к странице документации

Предусловие:

- создан JSON-файл с описанием виджета (widget.json);

Основной сценарий:

1. Пользователь открывает файл widget.json.

2. В строке указания типа поля в массиве fields отображается кнопка перехода.
3. Пользователь нажимает на кнопку.
4. В зависимости от типа поля, система открывает документацию на сайте Решения с описанием работы с полем.

3.1.2.4. Поиск и исправление ошибок

3.1.2.4.1. Проверка соответствия имени файла widget.json ключу name

Предусловие:

- создан JSON-файл с описанием виджета (widget.json);
- имя файла отличается от значения ключа name.

Основной сценарий:

1. Пользователь открывает файл widget.json.
2. Система подчеркивает ключ “name” и отображает иконку дополнительных функций.
3. Пользователь нажимает на иконку.
4. Система предлагает переименовать файл.
5. Пользователь подтверждает действие.
6. Система автоматически переименовывает файл в соответствии со значением ключа name.

3.1.2.4.2. Проверка соответствия имени файла view.json ключу name

Предусловие:

- создан JSON-файл с описанием представления (view.json);
- имя файла отличается от значения ключа name.

Основной сценарий:

1. Пользователь открывает файл view.json.
2. Система подчеркивает ключ “name” и отображает иконку дополнительных функций.
3. Пользователь нажимает на иконку.
4. Система предлагает переименовать файл.
5. Пользователь подтверждает действие.
6. Система автоматически переименовывает файл в соответствии со значением ключа name.

3.1.2.4.3. Проверка соответствия имени файла screen.json ключу name

Предусловие:

- создан JSON-файл с описанием экрана (screen.json);
- имя файла отличается от значения ключа name.

Основной сценарий:

1. Пользователь открывает файл screen.json.
2. Пользователь изменяет значение ключа name.
3. Система подчеркивает ключ “name” и отображает иконку дополнительных функций.
4. Пользователь нажимает на иконку.
5. Система предлагает переименовать файл.
6. Пользователь подтверждает действие.
7. Система автоматически переименовывает файл в соответствии со значением ключа name.

3.1.2.4.4. Проверка использования представления

Предусловие:

- созданы JSON-файлы с описанием представления и экрана (view.json и screen.json);
- имя представления указано в массиве views файла screen.json.

Основной сценарий:

1. Пользователь открывает файл widget.json.
2. Пользователь нажимает на иконку дополнительных функций в строке, где указан ключ widgetName.
3. Система проверяет, используется ли название виджета в других файлах проекта:
 - Если виджет нигде не используется, навигация не активируется, и система отображает соответствующее уведомление.
 - Если виджет используется, система предоставляет возможность навигации к файлам, где он упоминается.

3.1.2.4.5. Проверка использования виджета

Предусловие:

- созданы JSON-файлы с описанием виджета и представления (widget.json и view.json);
- имя виджета указано в массиве widgets файла view.json.

Основной сценарий:

1. Пользователь открывает файл widget.json.
2. Пользователь нажимает на иконку дополнительных функций в строке, где указан ключ widgetName.
3. Система проверяет, используется ли название виджета в других файлах проекта:

- Если виджет нигде не используется, навигация не активируется, и система отображает соответствующее уведомление.
- Если виджет используется, система предоставляет возможность навигации к файлам, где он упоминается.

3.1.2.4.6. Проверка ключа поля в виджете

Предусловие:

- создан JSON-файл с описанием виджета (widget.json);
- в ключе “key” указано значение, отличающееся от перечисленных значений в DTO файле.

Основной сценарий:

1. Пользователь открывает файл widget.json.
2. Система подчеркивает ключ “key” и отображает иконку дополнительных функций.
3. Пользователь нажимает на иконку.
4. Система предлагает создать новое поле.

3.1.2.4.7. Проверка на повторяющееся имя виджета (widgetName)

Предусловие:

- создан JSON-файл с описанием представления (view.json);
- в файле view.json в массиве views перечислены виджеты с одинаковым названием.

Основной сценарий:

1. Пользователь открывает файл view.json.
2. Система подчеркивает дублирующееся значение widgetName и отображает иконку дополнительных функций.
3. Пользователь нажимает на иконку.
4. Система предлагает удалить дубликаты.
5. Пользователь нажимает на кнопку.
6. Система удаляет все дубликаты и оставляет только один виджет с оригинальным значением widgetName.

3.1.2.4.8. Проверка на дублирование ключа на виджете

Предусловие:

- создан JSON-файл с описанием виджета (widget.json);
- в файле widget.json в массиве fields перечислены поля с дублирующимися ключами.

Основной сценарий:

1. Пользователь открывает файл widget.json.

2. Система подчеркивает дублирующийся ключ виджета и отображает иконку дополнительных функций.
3. Пользователь нажимает на иконку.
4. Система предлагает удалить дубликаты.
5. Пользователь нажимает на иконку.
6. Система удаляет все дубликаты и оставляет только одно поле с оригинальным значением “key”.

3.1.2.5. Инструменты мониторинга

3.1.2.5.1. Виды метрик инструментов мониторинга

В системе должны быть предусмотрены два типа метрик: стандартные и пользовательские. Стандартные метрики предоставляются с помощью Spring Boot Actuator, которые можно использовать для мониторинга состояния и производительности приложения (например, JVM Metrics, System Metrics, Tomcat Metrics, HTTP Metrics, Data Source Metrics). Пользовательские метрики специфичны для Системы и позволяют посмотреть количество запросов в разных разделах (например, статистика всех запросов к элементам UI, статистика запросов на проверку клиентов и пользователей)

3.1.2.5.2. Предусмотренные методы метрик

Для получения метрик в Системе должно быть реализовано API с методами запроса метрик.

Ответом каждого метода является файл JSON со значением метрики.

В API должны быть предусмотрены методы для следующих метрик:

- общее количество соединений;
- время установления соединения;
- количество активных соединений;
- время создания соединения;
- количество простаивающих соединений;
- максимальное количество соединений;
- минимальное количество соединений;
- количество ожидающих потоков;
- общее количество тайм-аутов соединений;
- время использования соединения;
- информация об http-запросах;

- текущее количество активных соединений, выделенных из источника данных JDBC;
- количество установленных, но простаивающих соединений JDBC;
- максимальное количество активных соединений JDBC, которые могут быть выделены одновременно;
- минимальное количество простаивающих соединений JDBC в пуле;
- оценка количества буферов в пуле;
- оценка размера памяти, которую виртуальная машина Java использует для данного буферного пула;
- оценка общей емкости буферов в данном пуле;
- количество классов, загруженных в данный момент в виртуальную машину Java;
- общее количество классов, выгруженных с момента начала работы виртуальной машины Java;
- размер пула памяти старого поколения после полной сборки мусора;
- максимальный размер пула памяти старого поколения;
- инкремент увеличения размера пула памяти нового поколения в период от одной сборки мусора до следующей;
- число положительных увеличений размера пула памяти старого поколения в период перед сборкой мусора и после сборки мусора;
- информация о сборке мусора после запуска данной виртуальной машины Java;
- объем памяти, который гарантированно доступен для использования виртуальной машиной Java;
- максимальный объем памяти, который можно использовать для управления памятью;
- объем памяти, используемый в данный момент;
- текущее количество активных демон-поточков;
- текущее количество активных потоков, включая демонов и не являющихся демонами;
- пиковое количество активных потоков с момента запуска виртуальной машины Java или сброса пика;
- текущее количество потоков в состоянии BLOCKED;
- количество событий уровня предупреждения, попавших в журналы;
- «недавняя загрузка процессора» процессом виртуальной машины Java;
- максимальное количество файловых дескрипторов;
- количество дескрипторов открытых файлов;
- unix-время начала процесса;
- время безотказной работы виртуальной машины Java;
- информация о вызовах репозитория;
- количество процессоров, доступных виртуальной машине Java;

- «недавняя загрузка процессора» всей системой;
- сумма количества работоспособных объектов, поставленных в очередь к доступным процессорам, и количества работоспособных объектов, работающих на доступных процессорах, усредненная за период времени;
- число активных сеансов Tomcat;
- максимальное количество сеансов Tomcat, которые были активными одновременно;
- самое длительное время, когда сеанс Tomcat с истекшим сроком действия был жив;
- количество созданных сеансов Tomcat;
- количество сеансов Tomcat с истекшим сроком действия;
- количество сеансов Tomcat, которые не были созданы, так как достигнуто максимальное количество активных сеансов;
- количество обращений к элементам ядра Решения.

3.1.2.5.3. Получение пользовательской метрики количества запросов к REST API

Предусловие:

- собрано приложение на основе CXBOX v6.0

Основной сценарий:

1. Пользователь выполняет запрос `.../actuator/metrics/platform-requests`;
2. Система предоставляет файл json со значением пользовательской метрики получения количества запросов к REST API backend, также показывая:
 - доступные экраны в приложении;
 - пользователей приложения;
 - crudmaAction (стандартные действия CRUD + пользовательские действия).

3.1.2.5.4. Получение пользовательской метрики количества запросов к REST API в разрезе экрана

Предусловие:

- собрано приложение на основе CXBOX v6.0

Основной сценарий:

1. Пользователь выполняет запрос следующего вида `.../actuator/metrics/platform-requests?tag=screen:<section value>`, где `:<section value>` название интересующего экрана (например, `.../actuator/metrics/platform-requests?tag=screen:client`);
2. Система предоставляет файл json со значением пользовательской метрики получения количества запросов к REST API по указанному экрану, также показывая:
 - пользователей, которым доступен указанный экран
 - сущности (bc), присутствующих на указанном экране

- список доступных crudmaAction (стандартные действия CRUD + пользовательские действия) на указанном экране.

3.1.2.5.5. Получение пользовательской метрики количества запросов к REST API в разрезе пользователя

Предусловие:

- собрано приложение на основе CXBOX v6.0

Основной сценарий:

1. Пользователь выполняет запрос следующего вида `.../actuator/metrics/platform-requests?tag=user:<section value>`, где `:<section value>` значение интересующего пользователя (например, `.../actuator/metrics/platform-requests?tag=user:DEMO`);
2. Система предоставляет файл json со значением пользовательской метрики получения количества запросов к REST API по указанному пользователю, также показывая:
 - сущности (bc), доступные указанному пользователю в приложении;
 - список экранов, доступный указанному пользователю в приложении;
 - список доступных crudmaAction (стандартные действия CRUD + пользовательские действия) указанному пользователю.

3.1.2.5.6. Получение пользовательской метрики количества запросов к REST API в разрезе пользовательского действия (custom action)

Предусловие:

- собрано приложение на основе CXBOX v6.0

Основной сценарий:

1. Пользователь выполняет запрос следующего вида `.../actuator/metrics/platform-requests?tag= crudmaAction:<section value>`, где `:<section value>` значение интересующего пользовательского действия (например, `.../actuator/metrics/platform-requests?tag= crudmaAction: sendEmail`);
2. Система предоставляет файл json со значением пользовательской метрики получения количества запросов к REST API по указанному пользовательскому действию, также показывая:
 - сущности (bc), для которых доступно указанное пользовательское действие;
 - список экранов, на которых присутствует указанное пользовательское действие;
 - список пользователей, которым доступно указанное пользовательское действие.

3.1.2.5.7. Получение иных пользовательских и стандартных метрик

Предусловие:

- собрано приложение на основе CXBOX v6.0

Основной сценарий:

1. Пользователь выполняет запрос следующего вида `.../actuator/metrics`;
2. Система предоставляет в формате json список доступных пользовательских и стандартных метрик;
3. Пользователь выбирает необходимую метрику;
4. Пользователь добавляет в запрос наименование метрики следующего вида `.../actuator/metrics/<metric names>`, где `<metric names>` наименование выбранной метрики
5. Система предоставляет файл json со значением выбранной метрики.

3.1.3. Системные дополнения

3.1.3.1. Виды хранилищ

Локальное хранилище представляет собой файловую систему сервера разрабатываемого приложения. S3-хранилище представляет собой объектное хранилище, предоставляемое облачным провайдером. Система должна обеспечивать возможность конфигурации приложения для хранения файлов в локальном хранилище или для использования инструментов подключения к S3-хранилищу выбранного провайдера.

Система должна обеспечивать возможность одновременной работы с несколькими хранилищами. Компоненты системы должны позволять программным путем выбирать хранилище, в котором следует сохранить или получить конкретный файл.

3.1.3.2. Сохранение и получение файла из хранилища

Для работы с файлами в хранилище в системе должен быть реализован интерфейс компонента сохранения и получения файла из хранилища. Компонент должен предоставлять возможности:

- проверки существования файла в хранилище;
- сохранения файла в хранилище;
- загрузки файла из хранилища;
- удаления файла из хранилища.

3.1.3.3. Загрузка файла на локальное устройство

Для загрузки файла на локальное устройство в системе должен быть реализован компонент, позволяющий загружать файлы из хранилища на локальное устройство.

3.1.3.4. Поле выбора файла

В системе должно быть реализовано поле экранной формы для выбора файла и сохранения его в хранилище. Компонент поля должен выполнять функции:

- открытие диалогового окна локальной файловой системы для выбора одного или нескольких файлов;
- сохранение выбранных файлов в файловое хранилище;
- предоставление ссылки на сохраненный файл.

Компонент должен предоставлять возможности настройки параметров процесса сохранения файла в хранилище:

- автоматическое сохранение файлов сразу после выбора или последующее программное управление сохранением;
- перетаскивание файла из файловой системы в окно браузера с последующим сохранением файла;
- использование горячей клавиши вставки с последующим сохранением файла;
- ограничение на размер сохраняемых файлов;
- перечень допустимых расширений файлов;
- отображение имени сохраненного файла рядом с полем;
- название, рисунок и описание кнопки выбора файла;
- название, рисунок и описание кнопки очистки поля.

Компонент должен предоставлять возможность обработки событий:

- значение поля очищено (перед очисткой);
- значение поля очищено (после очистки);
- значение поля изменилось;
- загрузка началась;
- загрузка завершилась;
- загрузка завершилась успешно;
- загрузка завершилась неудачно.

3.1.3.5. SSO авторизация

Система должна включать функцию авторизации пользователя с помощью SSO для использования в web-приложениях. Технология SSO представляет единый сервис аутентификации (провайдер) для подключения приложений с поддержкой таких протоколов, как OAuth (например, Keycloak)

3.1.3.6. Авторизация пользователя в приложении с помощью Keycloak

Предусловие:

- Система для аутентификации пользователя использует провайдера Keycloak;
- пользователь не авторизован в Системе

Основной сценарий:

1. Пользователь открывает web-приложение;
2. Система перенаправляет пользователя на страницу входа Keycloak;
3. Пользователь вводит корректные учетные данные (логин/пароль);
4. Пользователь нажимает кнопку «Sign In» (Войти);
5. Keycloak проверяет введенные учетные данные;
6. Keycloak перенаправляет пользователя обратно в web-приложение с кодом авторизации;
7. Система обменивает код авторизации на токены доступа и обновления;
8. Keycloak возвращает токены доступа и обновления;
9. Система сохраняет токены и создает сессию пользователю;
10. Пользователь авторизовывается в web-приложении.

3.1.3.7. Добавление пользователя в приложение с помощью Keycloak

Предусловие:

- на frontend приложения добавлена конфигурация по работе с keycloak;
- на backend приложения добавлена конфигурация по работе с keycloak;
- в application.yml установлено use-resource-role-mappings: true;
- администратор авторизован в клиенте провайдера.

Основной сценарий:

1. Открыть настройки провайдера keycloak;
2. Перейти в Administration Console;
3. Перейти в «Manage» -> «Users»;
4. Нажать на «Add user»;
5. Заполнить данные нового пользователя;

6. Перейти в «Role Mappings» нового пользователя;
7. В «Client roles» выбрать «схbox-oidc-client»;
8. Добавить для нового пользователя хотя бы одну роль;
9. Сохранить измененных конфигураций keycloak в git;
10. Перезапустить чистый контейнер keycloak;
11. Новый пользователь добавлен в приложение.

3.1.3.8. Добавление роли в приложение с помощью Keycloak

Предусловие:

- на frontend приложения добавлена конфигурация по работе с keycloak;
- на backend приложения добавлена конфигурация по работе с keycloak;
- в application.yml установлено use-resource-role-mappings: true;
- администратор авторизован в клиенте провайдера.

Основной сценарий:

1. Открыть настройки провайдера keycloak;
2. Перейти в Administration Console;
3. Перейти в «Configure» - «Clients»;
4. Выбрать «схbox-oidc-client» и перейти в «Roles»;
5. Нажать на «Add Role»;
6. Заполнить данные новой роли;
7. Добавить новую роль в словарь приложения, например в СХВОХ-DICTIONARY_ITEM.csv;
8. Сохранить измененных конфигураций keycloak в git;
9. Перезапустить чистый контейнер keycloak;
10. Новая роль добавлена в приложение.

3.1.4. Бизнес-дополнения

3.1.4.1. Получение пользователем push-уведомления

Предусловие:

- приложение поддерживает работу с WebSocket с помощью ActiveMQ

Основной сценарий:

1. Пользователь авторизовывается в приложении;
2. Система устанавливает WebSocket-соединение с сервером;

3. Сервер принимает запрос на установление WebSocket-соединения и устанавливает соединение с клиентом;
4. Сервер сохраняет соединение для отправки push-уведомлений в будущем;
5. В Системе событие, в том числе пользователь может его инициировать самостоятельно, требующее push-уведомление (например, отправка email-сообщения);
6. Сервер отправляет сообщение через WebSocket-соединение;
7. Приложение получает push-уведомление через установленное WebSocket-соединение;
8. Приложение отображает полученное push-уведомление пользователю в виде всплывающего окна в зависимости от backend инфраструктуры в одной вкладке браузера или всех, где пользователь авторизован.

3.1.4.2. Подключение приложения к push-уведомлениям используя ActiveMQ

Предусловие:

- приложение собрано на основе CXBOX v6.0

Основной сценарий:

1. Для websocket в pom.xml добавить зависимость spring-boot-starter-websocket;
2. Добавить новое окружение для websocket в application.yml;
3. Для ActiveMQ в pom.xml добавить новые зависимости: spring-boot-starter-activemq и spring-boot-starter-reactor-netty;
4. Преобразовать переменную среды ACTIVEMQ_BROKER_TYPE в значение «activemq»;
5. Указать другие параметры ActiveMQ, такие как: login, password, host, port, stomp_port;
6. Добавить конфигурации websocket на примере файла WebSocketConfig.java.

3.1.4.3. Подключение приложения к системе электронной почты

Предусловие:

- приложение собрано на основе CXBOX v6.0

Основной сценарий:

1. В pom.xml добавить зависимость spring-boot-starter-mail;
2. Добавить новое окружение в application.yml, где указать параметры подключения к системе электронной почты (host, port, username, password, использование SMTP-аутентификации, использование STARTTLS);
3. Добавить службу системы электронной почты по примеру файла MailSendingService.java

4. Указать параметры email-сообщений: получатель сообщения, тема сообщения, текст сообщения

3.1.4.4. Отправка пользователем email-сообщения из Системы

Предусловие:

- приложение собрано на основе CXBOX v6.0;
- приложение подключено к системе электронной почты.

Основной сценарий:

1. Пользователь открывает экран с action отправки email;
2. Пользователь инициирует отправку email-сообщения с помощью действия;
3. Приложение выводит пользователю сообщение, что email-сообщение отправляется;
4. Приложение обрабатывает запрос и отправляет email-сообщение на сервер отправки email;
5. Сервер отправки email отправляет email-сообщение;
6. Email-сообщение отправлено получателю;
7. Приложение отображает результат отправки email-сообщения для пользователя в окне уведомлений.

3.1.4.5. Установка предварительного просмотра для поля Файл

Предусловие:

- приложение собрано на основе CXBOX v6.0;

Основной сценарий:

1. Найти необходимый widget.json, где присутствует поле с типом fileUpload (Файл) и куда необходимо добавить предварительный просмотр;
2. Установить для поля настройку предварительного просмотра preview.enabled: true и preview.mode: popup;
3. Выполнить перезапуск приложения;
4. В интерфейсе приложения рядом с расширением загруженного файла (.pdf и картинки) отобразится значок глаза, говорящий, что файл можно открыть в модальном окне предварительного просмотра;
5. При нажатии на иконку расширения загруженного файла (с расширением .pdf, .png, .bmp, .gif, .ico, .cur, .jpg, .jpeg, .jif, .jpeg, .jpr, .png, .svg, .webp) будет открываться модальное окно предварительного просмотра;

3.1.4.6. Возможности модального окна предварительного просмотра

При нажатии на иконку предварительного просмотра Система должна открывать файл в модальном окне для предварительного просмотра. В окне предварительного просмотра должны быть доступны следующие возможности:

- Навигация: пользователь может переключаться между загруженными файлами (между файлами в той же колонке) с помощью кнопок “вперед” и “назад”. Порядок расположения файлов соответствует списку файлов в виджете и учитывает примененные фильтры и сортировку.
- Полноэкранный режим: файл открывается на весь экран в случае нажатия на кнопку “Fullscreen”. Возможность навигации в полноэкранном режиме сохраняется.
- Прокрутка страниц документа: для документов, состоящих из нескольких страниц, справа отображается полоса прокрутки, позволяющая пользователю переходить между страницами.
- Скачивание файла: в окне предпросмотра доступна кнопка “Download”, которая инициирует загрузку файла в папку “Загрузки” пользователя.

3.1.5. Дополнение администратора

3.1.5.1. Возможности расписания запуска заданий

Планировщик должен позволять задавать расписание запуска заданий со следующими возможностями:

- время суток (с точностью до миллисекунд);
- день недели;
- число месяца;
- день года;
- дни, указанные в специальном календаре;
- повтор заданное количество раз;
- повтор до заданной даты/времени;
- бесконечный повтор;
- повтор с заданным интервалом.

3.1.5.2. Запуск задания по расписанию

Предусловие:

- приложение собрано на основе CXBOX v6.0;

Основной сценарий:

1. В качестве задания может использоваться класс Java, реализующий соответствующий интерфейс;
2. Задания, назначенные к выполнению, создаются в статусе запланированного и отображаются в панели Администратора;
3. Система с заданной периодичностью просматривает список назначенных заданий во всех статусах и выполняет обработку в следующем порядке:
 - a. Если для задания наступило время запуска, то планировщик запускает задание на выполнение и переводит его в соответствующий статус, помещает задание в очередь на выполнение и отображает это в соответствующей вкладке в панели Администратора;
 - b. При наступлении очереди выполнения задания задание переходит в статус в работе;
 - c. Если задание успешно выполнено, то переводит его в статус успешного завершения и отображает это в соответствующей вкладке панели Администратора;
 - d. Если задание выполнено с отказом/ошибкой, то переводит его в статус завершения с отказом и отображает это в соответствующей вкладке панели Администратора.

Также администратор может удалить задание из любого статуса из панели Администратора, что переведет задание в статус удаленного и переместит задание в соответствующую вкладку на панели Администратора.

3.1.5.3. Статусы заданий

Система должна предусматривать следующие статусы заданий и соответствующие им вкладки в панели Администратора:

- запланировано – создано событие на выполнение задания, время выполнения задания еще не наступило;
- в очереди – задание запланировано к выполнению и находится в очереди выполняющихся заданий;
- в работе – задание выполняется;
- завершено успешно – задание было выполнено с успехом;
- завершено с отказом – задание было выполнено с ошибкой;
- удалено – задание было удалено Системой или Администратором.