**Практика KaspiLab**

**Цель**: Разработка и настройка прототипа Системы принятия решения.

**Требования к аудитории**:

1. Углубленные знания «ООП»;
2. Углубленные знания языка программирования «JAVA»;
3. Знание «JDK 8x»;
4. Углубленные знания языка «SQL»: DML и DDL;
5. Знания СУБД «MySQL 5x»;
6. Умение создавать web-приложения с использованием среды разработки «IntelliJ IDEA»;
7. Знания в реализации web-сервисов: REST, SOAP;
8. Знания форматов транспортных протоколов: JSON и XML;

**Результаты**:

1. Реализация процесса принятия решения на основе калькуляций, путем вызова через REST-сервис (end-point) и получение ответа.
2. Онлайн добавление новой калькуляции.
3. Онлайн изменение существующей калькуляции.
4. Отказоустойчивость программы.
5. Скорость выполнения до 1 сек.
6. **Термины и сокращения**

|  |  |
| --- | --- |
| **Система** | **Система принятия решения**. Web-приложение (back-end сервер), работающее в контейнере технологической платформы сервера приложений «Apache Tomcat 8x», публикующее синхронный REST-сервис (JSON), для межсистемного взаимодействия. |
| **БД** | База данных СУБД «MySQL 5x», обеспечивающая функциональные возможности хранения настроек системы и транзакционного журналирования хода исполнения процессов и калькуляций |
| **Калькуляция** | Формализованная часть алгоритма, как минимальная логическая единица (операция) процесса. Калькуляция, в большинстве случаев, представляет собой функцию обработки данных, со множеством входных параметров и одним выходным. Вычисленное калькуляцией значение, может являться входным значением для другой калькуляции, либо результирующим значением всего процесса. |
| **Тип калькуляции** | Условный классификатор, определяющий поведение калькуляции в процессе, а также пул таблиц, в которых калькуляция данного типа настраивается. В рамках системы должны быть реализованы следующие типы калькуляций:  - «**DESICISION\_TAB**» - Таблица решений (формализация конструкции программирования «Условный переход (IF-THEN-ELSE)»)  - «**GROOVY\_SCRIPT**» - скриптовое правило «GOOVY»;  - «**SQL\_SCRIPT**» - скриптовое правило «SQL»;  - «**REST\_SERVICE**» - вызов REST-сервиса (опционально). |
| **Процесс** | Линейный алгоритм принятия решения или набор калькуляций, выполняемых в заданной последовательности, с целью обработки входных данных, полученных от опубликованного «REST-сервис», и передаче внешней системе ответа о принятом решении. |
| **Вектор** | Внутренняя память процесса, в которой будут храниться входные данные процесса, промежуточные вычисления процесса, а также выходные данные процесса в формате JSON-Документа.  **ПРИМЕЧАНИЕ**: В целях однозначной идентификации и обеспечения функций логирования хода исполнения процесса, вектор и сам процесс должны быть поименованы уникальным идентификаторам «UUID». |
| **REST-сервис** | Опубликованный web-сервис системы, работающий в формате «POST»-запроса. В качестве формата обмена данными web-сервиса определяется строка «JSON» (Тип «TEXT/PLAIN»).  Для определения выполняемого процесса, системе должен быть передано значение кода процесса в качестве параметра заголовка REST-сервиса |

1. **Требования к организации БД**

**БД хранения настроек системы**:

1. Необходимо обеспечить функциональные возможности хранения данных о процессах, исполняемых калькуляциях в рамках каждого процесса, порядке исполнения каждой калькуляции процесса, а также флагов результатов исполнения каждого процесса. Рабочее название таблицы «**D\_PROC\_CALC**»;
2. Необходимо обеспечить функциональные возможности хранения данных о калькуляциях, их типов (**DESICISION\_TAB, GROOVY\_SCRIPT, SQL\_SCRIPT, REST\_SERVICE**), исполняемых скриптах (GROOVY, SQL). Рабочее название таблицы «**D\_CALC**»;
3. Необходимо обеспечить функциональные возможности хранения данных о всех параметрах процессов, их типах (**TEXT, NUMBER, DATE, TIMESTAMP, BOOLEAN**), а также JSON-Путь для записи и чтения данных в (из) вектор(а). Рабочее название таблицы «**D\_PAR**»;

**ПРИМЕЧАНИЕ**: Выходной параметр калькуляции определяется в таблице «D\_PAR» и соответствует ее кодовому имени.

1. Необходимо обеспечить функциональные возможности хранения данных таблицы решения (в рамках каждой калькуляции этого типа): приоритет условия, параметр сравнения, оператор сравнения (==, !=, >, <, >=, <=), сравниваемая константа, результат условия в рамках приоритета. Рабочее название таблицы «**D\_CALC\_DEC**»;

**БД логирования хода исполнения процесса**:

1. Необходимо обеспечить функциональные возможности хранения данных о входных и выходных данных процесса с указанием точного времени, когда входные данные были получены и точного времени, когда выходные данные были отданы во внешнюю систему. Рабочее название таблицы «ext\_log»;
2. Необходимо обеспечить функциональные возможности хранения данных о ходе исполнения процесса, а именно какие калькуляции какого процесса были исполнены с указанием точного времени исполнения. Рабочее название таблицы «proc\_log»;
3. Необходимо обеспечить функциональные возможности хранения данных, об ошибках сгенерированных в ходе исполнения процесса. Рабочее название таблицы «err\_log»;
4. **Функциональные требования**

**Краткое описание хода работы системы**:

1. Внешняя система вызывает опубликованный REST-сервис, передавая ему в качестве тела запроса JSON-строку и параметра заголовка – кодовое имя исполняемого процесса;
2. Система считывает код процесса и тело запроса, и осуществляет разбор JSON-строки, преобразуя его в вектор процесса;
3. Далее, на основании кода процесса, система считывает необходимые настройки процесса из БД и определяет порядок исполнения его калькуляций;
4. Далее, система последовательно считывает настройки каждой калькуляции процесса из соответствующих таблиц БД и интерпретирует ее исполнение, посредством соответствующего класса-обработчика;
5. На основании флагов результатов процесса, система подготавливает ответ во внешнюю систему – формирует JSON-строку;
6. Далее, система отдает ответ (сформированную JSON-строку) на запрос в опубликованный REST-сервис.

**ПРИМЕЧАНИЕ**: При реализации системы должно быть предусмотрено мульти-поточное исполнение двух и более процессов, неблокирующих друг друга.

**Общие требования к реализации**:

1. Web-приложение c использованием «Spring Framework»;
2. Сборка проекта «Maven 2x».

**Требования к реализации основных модулей системы**:

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Наименование** | **Краткое описание** | **Функциональные возможности** |
| 1 | REST Manager | REST-сервис для взаимодействия с внешними системами | В основе реализации должна быть заложено использование классов библиотеки «jersey (com.sun.jersey)»;  «End-point» для вызова исполняемого процесса из внешней системы.  **Тип запроса**: «POST».  **Медиа-тип**: «TEXT/PLAIN».  **Формат обмена данными**: строка «JSON»;  **Основной публикуемый метод**:  @POST @Path("/exec-proc") @Produces(MediaType.*TEXT\_PLAIN*) public String execProc(@HeaderParam("proc-code") String procCode, String json) throws Exception |
| 2 | Connection Manager | JDBC-соединения с БД из Connection Pool | **Способ исполнения**: «Singletone».  В основе реализации должна быть заложено использование класса «ComboPooledDataSource» библиотеки «c3p0 (com.mchange)»;  Настройки соединения с БД должны храниться в файле «db.properties», для переключения между различными средами (тестовая, боевая) без перекомпиляции приложения.  **Основной публикуемый метод**:  public Connection getConnection() |
| 3 | JSON Document | Внутренняя память процесса (вектор) | В основе реализации должна быть заложено использование классов библиотеки «jackson (com.fasterxml.jackson.\*)»;  Обеспечение функций разбора строки JSON, с возможностью преобразования в JSON-документ (serialization), чтения и сохранения данных по JSON-пути и генерации строки JSON (deserialization).  При чтении несуществующих значений процесс не должен «ломаться». Несуществующее значение должно интерпретироваться как пустое.  **Основные публикуемые методы**:  public void parse(String json) throws Exception  public String generate() throws Exception  public void setVal(String path, Object value) throws Exception  public Object getVal(String path) throws Exception |
| 4 | Process Manager | Процессор калькуляций | На основании кода процесса, должен выполнить чтение настроек из соответствующих таблиц БД и обеспечить последовательное исполнение калькуляций того или иного типа через соответствующие обработчики в заданном порядке исполнения.  **Основные публикуемые методы**:  public void execProc(String procCode) throws Exception |
| 5 | Decision Tab | Таблица решений | Обработчик (интерпретатор) калькуляции типа «**DECISION\_TAB**».  **ЛОГИКА ОБРАБОТКИ**: Внутри одного приоритета, условия должны сравниваться через логическую операцию «AND», между разными приоритетами через операцию «OR», т.е. если в рамках одного приоритета все условия («сработали») дали положительный результат (логическое значение «TRUE»), то дальнейшее рассмотрение условий по приоритету бессмысленно, и таблица решений должна вернуть результат, настроенный для этого приоритета. |
| 6 | Groovy Script | Скриптовое правило «GOOVY» | В основе реализации должна быть заложено использование классов «ScriptEngineManager» фабрики для «ScriptEngine».  В целях оптимизации использования Groovy-скриптов необходимо предусмотреть «singletone» оболочку для фабрики.  Обработчик (интерпретатор) калькуляции типа «**GROOVY\_SCRIPT**».  При настройке скрипта калькуляции все входные параметры должны быть указаны в формате «#PARAM\_CODE#», а при подготовке к исполнению должны быть найдены через регулярные выражения с шаблоном типа «#(.\*?)#» и привязаны к реальным значениям вектора.  Результирующее значение должно быть привязано к зарезервированному слову «RESULT».  **НАПРИМЕР**: RESULT = (#A#+#B#)\*#C#,  **ГДЕ**: #A#, #B# и #C# - входные параметры калькуляции, значения которых должны быть получены из вектора по их кодовым именам «A», «B» и «C» соответственно, а результирующее значение получено из переменной «RESULT» из записано в вектор по JSON-пути соответствующему кодовому имени калькуляции |
| 7 | SQL Script | Скриптовое правило «SQL» | Обработчик (интерпретатор) калькуляции типа «**SQL\_SCRIPT**».  В основе реализации должна быть заложено использование классов: «PreparedStatement», «ResultSet» и «ResultSetMetadata».  При настройке скрипта калькуляции все входные параметры должны быть указаны в формате «#PARAM\_CODE#», а при подготовке к исполнению должны быть найдены через регулярные выражения с шаблоном типа «#(.\*?)#» и привязаны к реальным значениям вектора  Должны поддерживаться следующе DML-операции работы с БД, которые не возвращают результирующее значение калькуляции:   * INSERT – вставка данных в таблицу   **ПРИМЕР**: «INSERT INTO T1(A,BC)VALUES(#A#,#B#,#C#)»;   * UPDATE – обновление данных в таблице   **ПРИМЕР**: «UPDATE T1 SET A=#A#,B=#B# WHERE C=#C#»;   * DELETE – удаление данных в таблице   **ПРИМЕР**: «DELETE FROM T1 WHERE A=#A# AND B=#B# AND C=#C#»,  а также операция «SELECT» в следующих режимах чтения данных из БД, определяющих тип курсора обработчика калькуляции:   * SIMPLE – чтение одного значения (например, агрегата)   **ПРИМЕР**: «SELECT COUNT(\*) AS CNT FROM T1»   * SINGLE – чтение одной записи   **ПРИМЕР**: «SELECT A, B, C FROM T1 WHERE ID = #ID#»   * MULTI – чтение нескольких записей   **ПРИМЕР**: «SELECT A, B, C FROM T1 WHERE ID IN (1,2,3)»  **ПРИМЕЧАНИЕ**: Вычитанные из БД данные должны быть сохранены в вектор по базовому JSON-пути, определяемому по кодовому имени калькуляции. При этом имена полей должны являться продолжением базового JSON-Пути (вложенными узлами для базового JSON-Пути), а в случае «MULTI» - массивом объектов. |

1. **Лабораторные работы**

**Лабораторная работа №1**. Настройка процесса принятия решения.

**Цель**: Настройка жестких правил проверки наличия физического лица в черном списке по установочным данным ИИН либо ФИО.

**Пример входных данных REST-сервиса**: {«IIN»:«823749498327», «FIRST\_NAME»:«ИВАН», «LAST\_NAME»:«ИВАНОВ», «LAST\_NAME»:«ИВАНОВИЧ»}

**Пример выходных данных REST-сервиса**: {«RESULT»:«Совпадений по ИИН или ФИО не найдено»}

**Ход исполнения лабораторной работы:**

1. В БД создаем прикладную таблицу со следующей структурой полей: «ИИН», «Фамилия», «Имя», «Отчество» и «Причина занесения в ЧС» с уникальными ключами по «ИИН» и «ФИО»;
2. Наполняем таблицу тестовыми данными;
3. Настраиваем калькуляцию «GROOVY\_SCRIPT» для проверки обязательности входящего значения «ИИН» **ИЛИ** группы параметров «ФИО», при отсутствии данных генерируем ошибку;
4. Настраиваем калькуляцию «SQL\_SCRIPT», которая возвращает «Причину занесения в ЧС» с поиском данных по «ИИН»;
5. Настраиваем калькуляцию «SQL\_SCRIPT», которая возвращает «Причину занесения в ЧС» с поиском данных по «ФИО»;
6. Настраиваем калькуляцию «DECISION\_TAB», которая проверяет наличие в векторе значения «Причина занесения в ЧС» по ИИН (результат: «Найдено совпадение по ИИН»), либо «Причина занесения в ЧС» по ФИО (результат: «Найдено совпадение по ФИО»), в противном случае результат: «Совпадений по ИИН или ФИО не найдено».
7. Добавляем калькуляции в процесс в вышеперечисленном порядке;
8. Флаг результата процесса выставляем на калькуляцию «DECISION\_TAB».
9. Проводим тестирование процесса с демонстрацией аудитории:

- Генерация ошибки при отсутствии входных данных;

- Получение ответа «Найдено совпадение по ИИН»;

- Получение ответа «Найдено совпадение по ФИО»;

- Получение ответа «Совпадений по ИИН или ФИО не найдено».

**Лабораторная работа №2**. Оптимизация процесса принятия решения.

1. Объединяем калькуляции «SQL\_SCRIPT», которые возвращают «Причину занесения в ЧС», таким образом, чтобы поиск осуществлялся по двум поисковым ключам;
2. Изменяем калькуляцию «DECISION\_TAB» для проверки одного выбранного из БД значения;
3. Проводим тестирование процесса с демонстрацией аудитории по вышеперечисленным кейсам.