МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

АВТОМАТИЗАЦИЯ СОЗДАНИЯ ВИТРИН РАНЖИРОВАННЫХ СПИСКОВ АБИТУРИЕНТОВ УНИВЕРСИТЕТА

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Мангасаряна Евгения Павловича

Научный руководитель
доцент, к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
доцент, к. ф.-м. н. _____ С. В. Миронов

Саратов 2025

СОДЕРЖАНИЕ

1 Технологии интеграции информационных систем

1.1 Протокол SOAP

SOAP представляет собой протокол обмена структурированными сообщениями, основанный на XML. Он обеспечивает стандартизированный, платформо-независимый обмен данными между приложениями. SOAP поддерживает строгую типизацию, обработку ошибок, расширяемость и высокий уровень безопасности через WS-Security, а также контрактную модель через WSDL.

Преимущества SOAP включают строгую схему данных и контрактное описание интерфейса, встроенные механизмы безопасности, надежную обработку ошибок и поддержку сложных транзакций, независимость от платформы и языка реализации. Несмотря на некоторые недостатки по сравнению с современными протоколами, надежность и простота настройки в «1С:Университет» делают SOAP основным протоколом интеграции с данной системой.

1.2 Управление программными зависимостями

В процессе разработки интеграционного программного комплекса особое значение имеет правильная организация управления внешними библиотеками и зависимостями. Это влияет на воспроизводимость среды, стабильность работы приложения и удобство дальнейшей разработки.

Poetry представляет собой специализированный инструмент, автоматизирующий все этапы работы с зависимостями и окружением Python-проекта. Все зависимости, их версии и метаданные проекта описываются в едином структурированном файле `pyproject.toml`. Poetry автоматически создаёт и управляет виртуальным окружением, формирует файл блокировки `poetry.lock` с точными версиями всех установленных зависимостей.

1.3 Программный интерфейс SOAP «1С:Университет»

Система «1С:Университет» предоставляет возможность создания SOAP API для интеграции с внешними сервисами, основанный на XML-форматировании данных и WSDL-контрактах. В рамках решения задач приёмной кампании системные администраторы университета сконфигурировали три ключевые конечные точки.

1.3.1 Операция `getSvodka`

Операция `getSvodka` предоставляет сводные данные о поступлении на различные специальности. Каждая запись в ответе содержит подробную информацию о конкретной специальности, включая её название, код, форму обучения, план приема, бюджетные и внебюджетные подачи, целевую организацию, конкурсную группу, факультет и специализацию. Эта операция имеет ключевое значение для составления общей картины приёма.

1.3.2 Операция `getSpisok`

Операция `getSpisok` предоставляет подробный список абитуриентов по каждой специальности. Каждая запись включает информацию о поступающем, специальности, форме обучения, форме оплаты, приоритете зачисления, баллах, а также о предоставлении оригиналов документов. Данная операция необходима для отслеживания отдельных заявок и обеспечения точного сбора всех подач абитуриентов по каждой конкурсной группе.

1.3.3 Операция `getExam`

Операция `getExam` предоставляет список вступительных испытаний и их приоритетность для каждой конкурсной группы. Каждая запись включает предмет испытания, заменяемые предметы и приоритет вступительного экзамена. Эта операция имеет важное значение для управления и проверки требований, предоставляя критерии соответствия абитуриентов конкурсным группам.

2 Инструменты разработки

2.1 Библиотека Zeep

Для реализации взаимодействия с SOAP API в Python используется пакет Zeep, обеспечивающий полный цикл работы с веб-сервисами. Zeep представляет собой современный SOAP-клиент с поддержкой протоколов SOAP 1.1/1.2, HTTP GET/POST, технологий безопасности WSSE и поддержкой asyncio для асинхронного выполнения запросов.

2.2 Архитектурные решения

Для ускорения выгрузки большого объёма данных применяется параллельная выгрузка записей. Поскольку каждый запрос может занимать неопределённое количество времени, через максимум 100 шагов интерпретации он высвобождает глобальную блокировку интерпретатора Python, позволяя другим потокам приступить к своим задачам.

Для контролируемого запуска параллельных вычислений используются пулы потоков, что позволяет значительно сократить общее время загрузки информации. Важно учитывать, что клиент Zeep не является потоко-безопасным, поэтому для каждого потока создаётся отдельный экземпляр класса `zeep.Client`.

Данные, получаемые из SOAP API, преобразуются в объектно-реляционные модели с помощью специальных адаптеров. Для обеспечения корректности и безопасности данных применяется валидация с использованием XSD-схем. Реализуется централизованная обработка исключений с фиксацией ошибок в журнале и возможностью повторных попыток обращения.

2.3 Организация хранения данных

2.3.1 MariaDB

Выбор MariaDB обусловлен её открытым исходным кодом, высокой производительностью при работе со связанными данными и полной совместимостью с MySQL-экосистемой. СУБД обеспечивает транзакционную обработку запросов, механизмы контроля целостности через foreign key, поддержку ACID-требований и индексирование для ускорения поиска.

Для работы со сложными связями данных в схеме реализована нормализация до 3NF, что исключает дублирование информации. Структура включает таблицы абитуриентов, конкурсных групп, экзаменов и их взаимосвязи.

2.3.2 Взаимодействие через объектно-реляционное отображение

Объектно-реляционное отображение позволяет работать с базой данных на уровне объектов языка программирования. Вместо ручного создания SQL-запросов разработчик описывает структуру данных в виде Python-классов, где каждый класс соответствует определённой таблице.

SQLAlchemy ORM предоставляет удобный и безопасный способ работы с реляционной базой данных. С помощью SQLAlchemy можно описывать связи между сущностями, автоматически следить за целостностью данных, корректно обрабатывать каскадные изменения и предотвращать SQL-инъекции.

2.4 Контейнеризация и автоматизация развёртывания

2.4.1 Применение Docker и Docker Compose

Docker предоставляет механизмы изоляции приложений через контейнеры, работающие на общем ядре ОС с собственными зависимостями и окружением. Основные преимущества включают слоистую файловую систему, управление ресурсами и сетевую изоляцию.

Docker Compose расширяет функциональность, позволяя описывать многоконтейнерные приложения в YAML-манифесте. Такая конфигурация обеспечивает автоматическое разрешение доменных имён сервисов, изоляцию групп контейнеров через отдельные сети и централизованное управление переменными окружения.

2.4.2 MariaDB для разработки

Для разработки создаётся контейнер СУБД MariaDB с использованием Docker Compose. Такой подход позволяет быстро поднимать изолированную среду, максимально приближенную к рабочему окружению, не влияющую на реальные данные.

Важной частью настройки является автоматическая инициализация схемы базы данных через файлы в `/docker-entrypoint-initdb.d/`, которые выполняются при первом запуске MariaDB. Это обеспечивает быструю подготовку среды без ручного создания таблиц.

3 Реализация интеграционного решения

3.1 UML-диаграмма последовательности

Были построены две UML-диаграммы последовательности, отражающие основные сценарии загрузки и обновления данных о поступающих в университет и колледжи. Диаграммы формализуют логику взаимодействия между основными компонентами системы и определяют ключевые точки параллелизма при работе с внешним SOAP API и внутренней базой данных MariaDB.

Процесс загрузчика университета включает запуск загрузчика, загрузку конфигурации, создание сессии соединения с базой данных, каскадную очистку таблиц, загрузку данных 1С «Svodka», заполнение справочников, параллельную загрузку вступительных испытаний и списков абитуриентов, формирование витрин и завершение работы.

3.2 Структурная организация и модули приложения

Логика проекта организована в пакете `admission`, который объединяет все ключевые элементы решения. На верхнем уровне располагаются модули `loader` и `loader_coll` для загрузки данных университета и колледжа, функция инициализации журналирования, набор моделей конфигурации и модуль обработки аргументов командной строки.

Модуль `utils` содержит вспомогательные функции для преобразования форм обучения и наименований конкурсных групп. Модуль `db` отвечает за взаимодействие с реляционной базой данных и включает определения моделей SQLAlchemy, загрузчики данных, вспомогательные процедуры для работы с базой данных и модули построения витринных таблиц.

Модуль `wsdl` реализует интеграцию с SOAP API «1С:Университет» и содержит средства для подключения к внешним сервисам, функции многопоточной загрузки данных и обработчики конкретных задач.

3.3 Конфигурирование интеграционного решения

3.3.1 Формат хранения конфигурации

Параметры конфигурации хранятся во внешнем файле в формате JSON. Такой подход обладает рядом преимуществ: JSON является широко распространённым структурированным форматом, в одном файле могут содержаться параметры для различных сред, формат хорошо поддерживается большинством языков программирования.

Каждый объект верхнего уровня описывает отдельный контур: параметры подключения к серверу 1С, реквизиты базы данных для записи информации, уровень очистки при подготовке таблиц к выгрузке и другие необходимые атрибуты.

3.3.2 Модель и обработка конфигурации

Для надёжного и типобезопасного чтения, валидации и использования параметров конфигурации применяется библиотека Pydantic. Она позволяет описывать структуру данных с помощью классов Python, автоматически проводить валидацию и преобразование типов.

Реализована иерархия моделей Pydantic, полностью отражающая структуру конфигурационного файла. Для загрузки настроек используется функция `get_config`, которая по переданному аргументу выбирает необходимый блок конфигурации, валидирует параметры и возвращает экземпляр модели `ConfigData`.

3.4 Организация журналирования

Модуль `logging` отвечает за настройку журнала событий и является неотъемлемой частью надёжной системы. Журналирование позволяет отслеживать ход выполнения процессов, фиксировать ошибки, предупреждать о сбоях и проводить последующий анализ работы приложения.

Модуль использует стандартную библиотеку Python logging и расширение `TimedRotatingFileHandler` для ротации журналов по времени. Основная функция `init_logger` инициализирует параметры журналирования для всего приложения.

Для организации вывода сообщений используются два обработчика: `TimedRotatingFileHandler` сохраняет журнал в файл с ежедневной ротацией и хранит до 14 архивных файлов, `StreamHandler` выводит сообщения в стандартный поток вывода. Важной особенностью является автоматическое создание директории для журналов и гибкая настройка уровня журналирования.

3.5 Объектно-реляционные модели

SQLAlchemy позволяет описывать структуру таблиц с помощью классов, что значительно облегчает работу с данными и повышает читаемость кода. Для

создания моделей используется базовый класс, создаваемый с помощью вызова функции `declarative_base`.

При описании модели необходимо явно указать имя таблицы, её столбцы и дополнительные параметры. Структура таблицы уже существует и не подлежит изменению, поэтому необходимо строго следовать её определению для обеспечения корректного сопоставления между моделью и таблицей.

Пример модели для справочника подразделений включает класс `Dep`, наследующийся от базового класса `SSUAbit`, с параметрами `__tablename__` для определения имени таблицы, `__table_args__` для общего комментария и атрибутами для описания столбцов с соответствующими типами данных и ограничениями.

3.6 Организация обновления данных с минимальным временем простоя

В ситуациях, когда база данных содержит множество таблиц, используемых разными приложениями, важно обновлять только необходимые объекты, не затрагивая остальные данные и не нарушая работу других сервисов. Одним из эффективных способов минимизации времени простоя при обновлении данных является использование временных таблиц с последующей заменой основных таблиц на новые.

Для реализации такого подхода сначала создаются временные таблицы с нужной структурой и именами, отличающимися от рабочих. В эти временные таблицы загружаются актуальные данные, выполняются все необходимые проверки и подготовительные операции. При этом основные таблицы продолжают обслуживать текущие запросы пользователей и приложений.

После завершения загрузки и проверки данных наступает этап замены. Для этого используются атомарные операции переименования таблиц, которые поддерживаются большинством современных систем управления базами данных. В момент переключения основная таблица переименовывается с добавлением специального суффикса `_old`, а временная таблица получает имя рабочей.

Такая операция выполняется очень быстро, поскольку затрагивает только метаданные базы данных и не требует физического копирования данных. Это позволяет сократить время недоступности информации до минимума. Данный способ особенно актуален для систем, где критично минимальное время простоя

и требуется обновить только часть таблиц, не затрагивая остальные объекты базы данных.

3.7 Организация многопоточного взаимодействия с SOAP API

В рамках интеграционного решения реализована задача массовой выгрузки сведений о вступительных испытаниях и абитуриентов для конкурсных групп из SOAP API «1С:Университет». Поскольку сервер поддерживает параллельную обработку запросов и подобная операция повторяется для каждой группы, применяется модель параллельного запуска фиксированного числа потоков.

Каждый поток получает индивидуальную задачу из очереди. Такой подход базируется на классической схеме «поток-обработчик» (consumer pattern), что позволяет существенно сократить общее время интеграционного этапа.

3.7.1 Механизм получения потокобезопасного клиента

Ключевой особенностью взаимодействия с SOAP API через библиотеку Zeep является требование потоковой изоляции клиентов: экземпляр `zeep.Client` не является потокобезопасным и должен быть уникальным для каждого потока. Для решения этой задачи используется специальный декоратор, обеспечивающий корректное распределение SOAP-клиентов между потоками.

Для каждого потока в объекте, локальном по отношении к нему (`threading.local()`), хранится собственный экземпляр клиента. Декоратор автоматически обеспечивает инициализацию клиента только при первом обращении в данном потоке, что предотвращает конфликты доступа к общему состоянию клиента и гарантирует потоковую безопасность.

3.7.2 Функция многопоточной выгрузки данных

Функция `exams_unload_from_1C` отвечает за организацию многопоточной выгрузки результатов вступительных испытаний. Логика функции составлена из следующих шагов: инициализация подключения на основании настроек конфигурации, формирование очереди заданий для каждой конкурсной группы в общую потокобезопасную очередь, запуск потоков-обработчиков с количеством, определяемым конфигурационным параметром, и итоговая запись результатов после завершения всех потоков.

В список результатов от каждого потока дописываются объекты моделей экзаменов после обработки ответа SOAP API. Все потоки запускаются одновременно и далее синхронизируются по завершению через `join()`. После завершения всех потоков накопленный список экзаменов сохраняется в базу данных через функцию массовой вставки.

3.8 Контейнеризация и сборка приложения

3.8.1 Организация среды с Docker Compose

Для запуска всех необходимых компонентов локально применяется Docker Compose — инструмент, позволяющий описывать мульти-контейнерную архитектуру в едином YAML-манифесте. В проекте есть возможность одновременно запустить два сервиса: контейнер `mariadb` для хранения витрин и контейнер `app` с интеграционным приложением.

Для хранения данных MariaDB используется отдельный том, расположенный по пути `./mariadb/data`. В каталог `/docker-entrypoint-initdb.d` монтируется директория `./mariadb/init`, содержащая необходимые SQL-сценарии. Это обеспечивает автоматическую инициализацию структуры базы данных и её наполнение при первом запуске контейнера.

Благодаря такому подходу нет необходимости вручную создавать таблицы и заполнять их начальными данными. Контейнер MariaDB настроен на автоматический перезапуск, что способствует надёжной и устойчивой работе среды разработки.

3.8.2 Сборка образа загрузчика

Процесс сборки образа контейнера приложения реализован через Dockerfile с несколькими стадиями, обеспечивая компактность образа, эффективную повторную сборку и воспроизводимость среды. На первом этапе используется полноценный базовый образ Python 3.11 для компиляции зависимостей с установкой необходимых системных библиотек и Poetry нужной версии.

На втором этапе используется компактный образ Python 3.11 с размещением виртуального окружения, собранного на предыдущем этапе, а также исходного кода, конфигурации и стартового сценария. Переменная `PATH` указывает на директорию виртуального окружения, а в качестве точки входа указывается запуск сценария `entrypoint.sh`.

Выбранный подход к сборке контейнера обеспечивает существенное уменьшение размера итогового образа, высокую воспроизводимость результата благодаря фиксированным версиям библиотек и наличию файла блокировки зависимостей, а также необходимую гибкость для добавления переменных окружения и вспомогательных сценариев.

4 Тестирование и результаты работы программы

В рамках тестирования разработанного программного комплекса была произведена серия запусков на специально подготовленных тестовых данных. В тестовом наборе содержалось 18 подач, причём структура данных включала разнообразные конкурсные группы, что позволяет считать тестовую выборку репрезентативной для оценки производительности и корректности работы системы.

В ходе испытаний полное время работы программы на указанном объёме данных составило 2 минуты и 14 секунд. Следует отметить, что время выполнения в данном случае определяется не количеством подач, а количеством конкурсных групп, поскольку для каждой из них осуществляется отдельная параллельная выгрузка и обработка данных.

На момент проведения испытаний приёмная кампания ещё не была запущена, поэтому полноценное тестирование в условиях реальной нагрузки будет проведено позднее. Тем не менее, текущие результаты позволяют судить о работоспособности и стабильности решения на типовых сценариях, максимально приближённых к эксплуатационным условиям.

Полученное время выполнения можно считать приемлемым для задач автоматизации интеграции данных между внутренними системами университета и внешними сервисами. Программа успешно справилась с обработкой всех конкурсных групп, не допустив сбоев или потери данных. Учитывая, что в реальной эксплуатации объём подач может быть существенно выше, архитектура решения предусматривает масштабирование за счёт многопоточной обработки и эффективного взаимодействия с внешними сервисами.