ГЕНЕРАЦИЯ ПРОГРАММНОГО КОДА СЛОЯ ДОСТУПА К ДАННЫМ И НАБОРОВ ТЕСТОВЫХ ДАННЫХ ДЛЯ ПРИЛОЖЕНИЙ, ИСПОЛЬЗУЮЩИХ РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ

В. Н. Король

Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», Республика Беларусь

Научный руководитель О. Д. Асенчик

Разработан программный комплекс, автоматизирующий создание кода слоя доступа к данным и генерацию осмысленных тестовых данных. Отмечено, что разработанный графический интерфейс упрощает проектирование структуры хранилища данных, а специальный алгоритм обеспечивает корректную генерацию кода и тестовых данных с учетом связей и бизнес-логики. Выделено, что это позволяет ускорить разработку, минимизировать ошибки и упростить поддержку программного комплекса.

Ключевые слова: проектирование баз данных, генерация тестовых данных, кодогенерация, языковые модели, слой доступа к данным.

Одной из ключевых проблем традиционного подхода к созданию слоя доступа к данным (*DAL*), лежащих в основе большинства приложений баз данных, является их высокая трудозатратность. Разработчикам приходится вручную писать повторяющийся код для реализации типовых *CRUD*-операций, что не только замедляет процесс разработки, но и увеличивает вероятность ошибок. Человеческий фактор может привести к некорректным *SQL*-запросам, неоптимальному выполнению операций и усложнению поддержки кода при изменениях в структуре базы данных. Существующие решения, такие как *ORM*-фреймворки (*Entity Framework, Hibernate, SQLAlchemy*), значительно упрощают работу с базами данных, но требуют написания программного кода моделей и зачастую не обеспечивают оптимальную генерацию *SQL*-запросов [1]. Кроме того, кодогенераторы, такие как *Scaffold* в *EF Core, CodeSmith* и *T4 Templates*, предоставляют лишь шаблонные решения, которые необходимо адаптировать под конкретные бизнес-требования.

Еще одной важной проблемой является генерация тестовых данных. Существующие инструменты, такие как *Faker, Bogus* и *SQL Data Generator*, которые позволяют автоматически наполнять базы данных случайными значениями, но не учитывают связи между таблицами и бизнес-логику приложения. В результате тестовые данные могут быть неполноценными, что снижает их полезность при тестировании системы.

Предлагаемый программный комплекс решает эти проблемы за счет интеграции визуального проектирования базы данных и интеллектуальной генерации кода *DAL*. В системе реализован удобный графический интерфейс, который позволяет пользователям создавать и настраивать структуру базы данных без необходимости писать *SQL*-код вручную. На основе созданной модели автоматически генерируется *SQL*-скрипт, который затем используется для создания базы данных. На основе сгенерированного *SQL*-скрипта система формирует слой доступа к данным с учетом выбранного языка программирования (*C#*, *Python* или *JavaScript*), *ORM* и архитектурного подхода. Для этого используются специализированные инструменты, характерные для каждой технологии. Например, в случае *Entity Framework* применяется *Scaffold-DbContext*, который позволяет автоматически создавать модели и контекст базы данных, обеспечивая удобную работу с данными на уровне кода [2].

Одним из ключевых элементов разрабатываемого программного комплекса является возможность интеллектуальной генерации тестовых данных. В отличие от традиционных методов, основанных на случайной генерации значений, система использует большие языковые модели (*LLM*), что позволяет формировать осмысленные и логически взаимосвязанные тестовые наборы, максимально приближенные к реальным данным. Процесс генерации тестовых данных реализован в виде распределенной системы, состоящей из нескольких ключевых компонентов.

Клиентская часть приложения разработана на *React* и представляет собой удобный и интуитивно понятный интерфейс для взаимодействия пользователя с системой. Одним из ключевых компонентов интерфейса является визуальный конструктор базы данных, который позволяет моделировать структуру данных, создавать таблицы, определять связи между ними и задавать параметры полей. Конструктор поддерживает настройку типов данных, ограничений и зависимостей, что делает процесс моделирования более наглядным.

Серверная часть приложения построена на платформе *ASP.NET Web API*, обеспечивая взаимодействие между клиентским интерфейсом, системой управления базами данных и другими внешними сервисами. Данный сервис содержит основную часть бизнес-логики. Например, компонент для генерации *SQL*-скриптов реализумый при помощи *Scriban* – мощного шаблонизатора, позволяющего формировать код в зависимости от структуры базы данных и заданных параметров [3]. *ASP.NET Web API* служит связующим звеном между клиентской частью, реализованной на *React*, и внутренними сервисами системы. При получении запроса сервер анализирует его, выполняет необходимые вычисления или обработку данных и формирует ответ, который затем передается обратно клиенту.

В качестве системы управления базами данных используется *PostgreSQL*, которая предоставляет надежные механизмы хранения и обработки данных, а также поддерживает сложные транзакционные операции.

Ключевым компонентом системы, отвечающим за интеллектуальную генерацию тестовых данных, является специализированный сервис, разработанный на *Python*. Он играет центральную роль в распределении нагрузки и организации параллельной обработки данных, обеспечивая высокую производительность процесса генерации тестовых данных.

Структурно данный сервис можно разделить на две основные части. Первая – это управляющий сервис, который принимает запросы на генерацию тестовых данных от *ASP.NET Web API*. Получив запрос, он анализирует его, определяет структуру базы данных, учитывает связи между таблицами и бизнес-правила, а затем разбивает задачу на несколько подзадач. Эти подзадачи затем распределяются между кластерами.

Вторая часть системы – это кластеры, представляющие собой отдельные *Python*-приложения. Каждый кластер отвечает за генерацию тестовых данных для конкретной таблицы или группы таблиц. В рамках своей работы он формирует корректные промты для *ChatGPT API*, отправляет запросы, получает сгенерированные данные и приводит их к требуемому формату.

Для лучшего понимания работы приложения рассмотрим процесс взаимодействия всех его компонентов.

На первом этапе пользователь формирует структуру базы данных в веб-конструкторе на *React*. Далее для получившейся базы данных отправляется запрос на генерацию тестовых данных к *ASP.NET Web API* серверу. Далее основной сервис отправляет запрос на генерацию тестовых данных сервису на *Python*. В запросе

передается информация о структуре базы данных, включая список таблиц, их связи, типы данных в колонках, ограничения и бизнес-правила, если они заданы. Принятый запрос обрабатывается сервером управления задачами, который анализирует его, разбивает на подзадачи и распределяет их между кластерами.

Для генерации данных в каждом кластере формируется промт (запрос) к *ChatGPT API*. Промты создаются с учетом структуры таблицы. Например, если в базе данных есть таблица пользователей, промт может содержать инструкцию: «Сгенерируй 100 записей для таблицы *user*. Поля: *name* (имя), *email* (электронная почта), *age* (возраст). Учитывай, что *email*-адреса должен быть уникальным.»

После формирования промта кластер отправляет запрос к *ChatGPT API*, получает ответ в виде сгенерированных данных и приводит данные к нужному формату. Когда все кластеры завершают обработку своих частей, сервер управления задачами объединяет результаты и передает их обратно основному сервису через *API*. На этом этапе формируется итоговый *SQL*-скрипт для вставки данных в базу, который затем передается пользователю. Схема взаимодействия компонентов системы представлена на рис. 1.

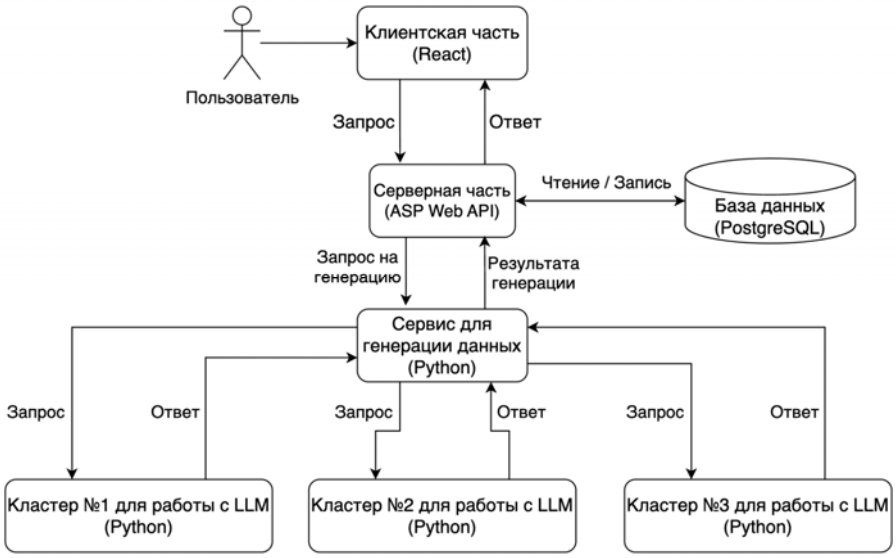

Рис. 1. Схема взаимодействия приложения

Разработнанный программный комплекс *DbDesigner* [4] представляет собой комплекс для автоматизированного создания *DAL* программного решения, а также интеллектуальной генерации наборов тестовых данных. Основной его особенностью является удобный графический интерфейс, который позволяет пользователям визуально проектировать структуру базы данных, включая таблицы, поля и связи между ними. Это упрощает процесс моделирования и снижает порог вхождения для разработчиков, давая возможность сосредоточиться на бизнес-логике приложения, а не на ручном написании *SQL*-запросов и конфигурации *ORM*. Также он упрощает тестирование за счет интеллектуальной генерации осмысленных данных.

Таким образом, удобное визуальное моделирование, автоматическая генерация кода и механизмы формирования тестовых данных делают систему универсальной и применимой для различных предметных областей, обеспечивая разработчиков мощным инструментом для работы с реляционными базами данных.

Литература

1. Object–relational mapping / Wikipedia. – URL: https://en.wikipedia.org/wiki/Object–relational_mapping (дата обращения: 10.02.2025).
2. Entity Framework Core / Microsoft Docs. – URL: https://learn.microsoft.com/en-us/ef/core/ (дата обращения: 10.02.2025).
3. Scriban / GitHub. – URL: https://github.com/scriban/scriban (дата обращения: 10.02.2025).
4. DbDesigner / GitHub. – URL: https://github.com/Javaro3/DbDesigner (дата обращения: 10.02.2025).

КОМПЬЮТЕРНАЯ СИСТЕМА ОБУЧЕНИЯ КИТАЙСКОМУ ЯЗЫКУ С ПРИМЕНЕНИЕМ ГОЛОСОВОГО ИНТЕРФЕЙСА

С. С. Эзрин

Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», Республика Беларусь

Научный руководитель В. С. Мурашко

Рассмотрена необходимость создания компьютерной системы обучения китайскому языку с применением голосового интерфейса. Определены задачи приложения и описана трехуровневая архитектура приложения, обеспечивающая его надежность и масштабируемость.

Ключевые слова: пользователь, прогресс пользователя, уроки, занятия, *HSK*-уровни, правила, тэги, трехслойная архитектура.

Обучение китайскому языку представляет собой сложный процесс, включающий освоение фонетики, лексики, грамматики и письменности. В традиционном обучении широко используются учебники, аудиокурсы и занятия с преподавателем. Однако современные технологии позволяют значительно расширить возможности изучения языка, включая использование компьютерных систем и голосовых интерфейсов.

Основные недостатки большинства подобных решений заключаются в отсутствии эффективного контроля правильности произношения и недостаточной адаптивности к уровню знаний пользователя.

Целью данной работы является разработка программного обеспечения, которое предоставит пользователю возможность эффективно изучать китайский язык, сочетая хранение словаря и аудио на локальном устройстве с использованием серверной базы данных для авторизации и сохранения прогресса. Основные требования к системе – это высокая точность распознавания речи, качество синтеза речи, удобство интерфейса и адаптивность к различным платформам.

Данная система состоит из клиентской и серверной частей, которые выполняют различные задачи, связанные с обучением китайскому языку, распознаванием речи и синтезом речи.

Обучающая система китайскому языку моделируется для одного актора – ученика. Основные бизнес-процессы, которые выделяются для него: прохождение регистрации/авторизации; указание и изменение уровня владения языком; прохождение урока, который делится на теоретическую и практическую части; просмотр изученных слов, а также изученных правил.