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

СИСТЕМА ХРАНЕНИЯ И ГЕНЕРАЦИИ УЧЕБНЫХ ТЕСТОВ В ФОРМАТЕ SCORM

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Ковешникова Даниила Вадимовича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

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

Саратов 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На данный момент в сфере образования существуют разные образовательные платформы, содержащих большое количество учебных тестов, которые являются частичной или полной копией друг-друга. При этом часть этих повторяющихся тестов создается одним и тем-же преподавателем из-за отсутствия общего хранилища данных материалов, что добавляет лишнюю нагрузку на сотрудника. В связи с вышеизложенным актуальность работы определяется необходимостью реализации приложения, позволяющего уменьшить нагрузку на преподавателей путем создания общего хранилища тестовых материалов, а также реализации методов интеграции их в образовательные платформы.

Целью данной работы является реализация микросервисной системы хранения, генерации в формате SCORM и предоставления учебных тестов для разных учебных платформ. В процессе выполнения данной работы необходимо решить следующие задачи:

- создание архитектуры разрабатываемой системы;
- выбор способа и построение модели хранения данных;
- реализация механизма генерации SCORM-пакетов;
- реализация интерфейса доступа к данным;

1 Анализ предметной области

1.1 scorm-пакет

Scorm-пакет представляет собой *.zip* архив, составленный в соответствии со спецификацией SCORM. SCORM является стандартом, применяемым при создании и передаче курсов. В данный момент SCORM является одним из наиболее популярных форматов представления курсов, используемых при обучении.

Спецификация SCORM (Sharable Content Object Reference Model) — стандарт, используемый для создания и распространения образовательных материалов. Основная цель данной спецификации состоит в идее создания курса единожды и использовании на любой образовательной платформе, поддерживающей данный стандарт. Это снижает нагрузку на преподавателей и организации, и позволяет переносить учебные материалы между платформами, без необходимости повторного создания.

1.1.1 Структура

SCORM-пакет состоит из следующего набора объектов:

— `imsmanifest.xml` — файл конфигурации scorm-пакета, содержащий информацию об используемой схеме пакета, названии пакета, организации и используемых файлах в пакете;
— директория `resources`, хранящая в себе все материалы, используемые в пакете;
— `.html` файлы, содержащие образовательный контент;
— служебные файлы.

1.2 Архитектура программного обеспечения

Архитектурой ПО(программного обеспечения) называется набор решений об организации программной системы, состоящий из:

— выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;
— способы взаимодействия и поведения выбранных элементов в реализуемом обеспечении;
— стиль реализации всех элементов обеспечения.

Существуют следующие наиболее популярные виды архитектур: монолитная и микросервисная.

1.2.1 Монолитная архитектура

Данная модель является традиционной при разработке программного обеспечения. Она представляет единый модуль, работающий независимо от других приложений. Данная архитектура представляет собой единую сеть с одной базой кода, в которой объединены реализации всех требуемых задач.

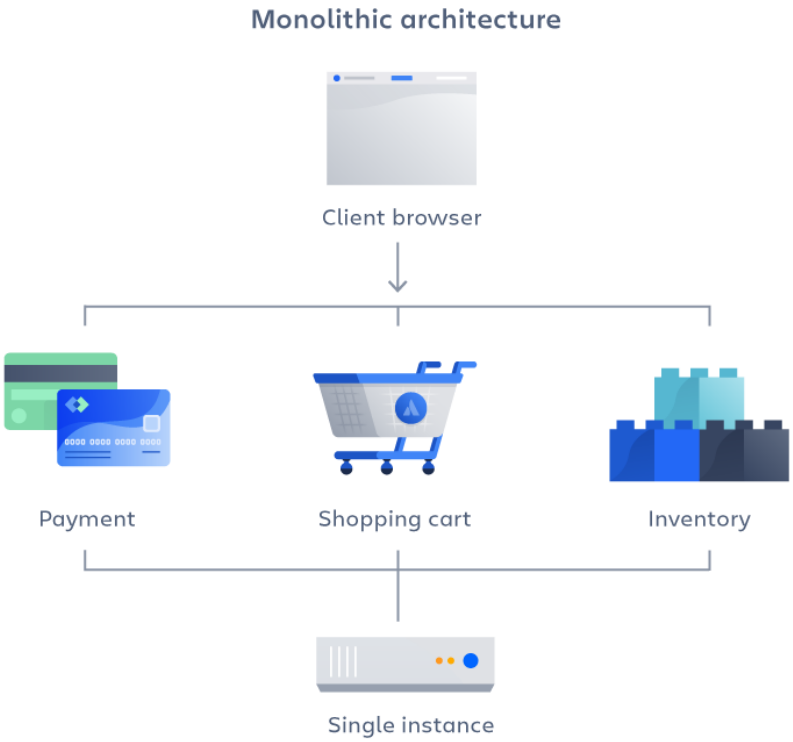

Рисунок 1.1 – Внешний вид монолитной архитектуры

1.2.2 Микросервисная архитектура

Данная модель представляет собой метод организации системы, основанный на ряде независимых служб. У данных служб существует собственная логика работы и конкретная цель. Все обновления развертывания и масштабирования происходят для каждой службы отдельно. Данная модель разбивает крупные задачи на несколько независимых, реализуемых в разных базах кода. Благодаря этому разделению задач любые сложности процесса становятся более видимыми и управляемыми.

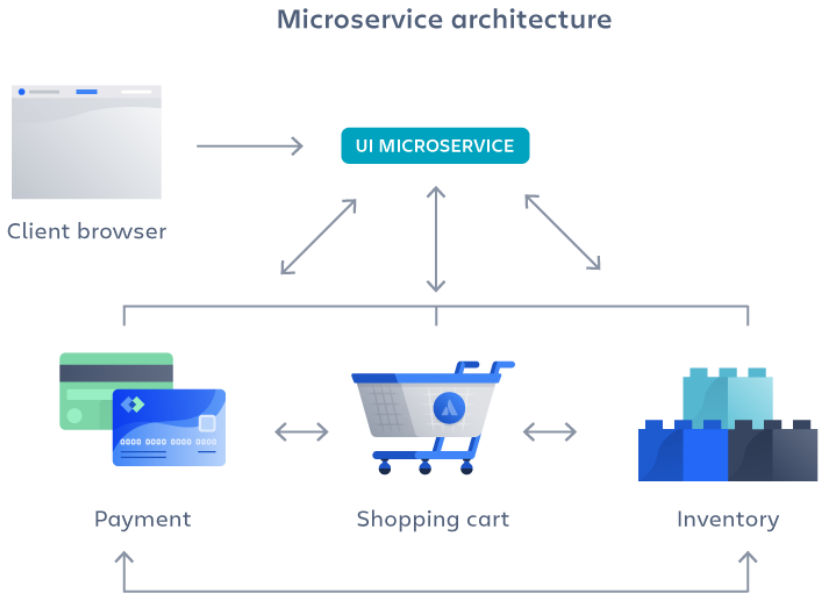

Рисунок 1.2 – Внешний вид микросервисной архитектуры

1.2.3 UML

Для удобства разработки архитектуры ПО был создан язык UML. Данный язык был разработан с целью обеспечить единый визуальный язык с богатой семантикой и развернутым синтаксисом для проектирования и внедрения программных систем.

В uml используется набор объектов, соединяемых между собой различными способами с целью создания схем, которые отражают различные аспекты системы.

1.3 Docker

Docker представляет собой открытую платформу для разработки и использования программного обеспечения. Она позволяет упаковывать ПО в контейнер вместе с его окружением и зависимостями, изолируя его от других приложений и операционной системы. Его архитектура предоставлена на изображении 1.3:

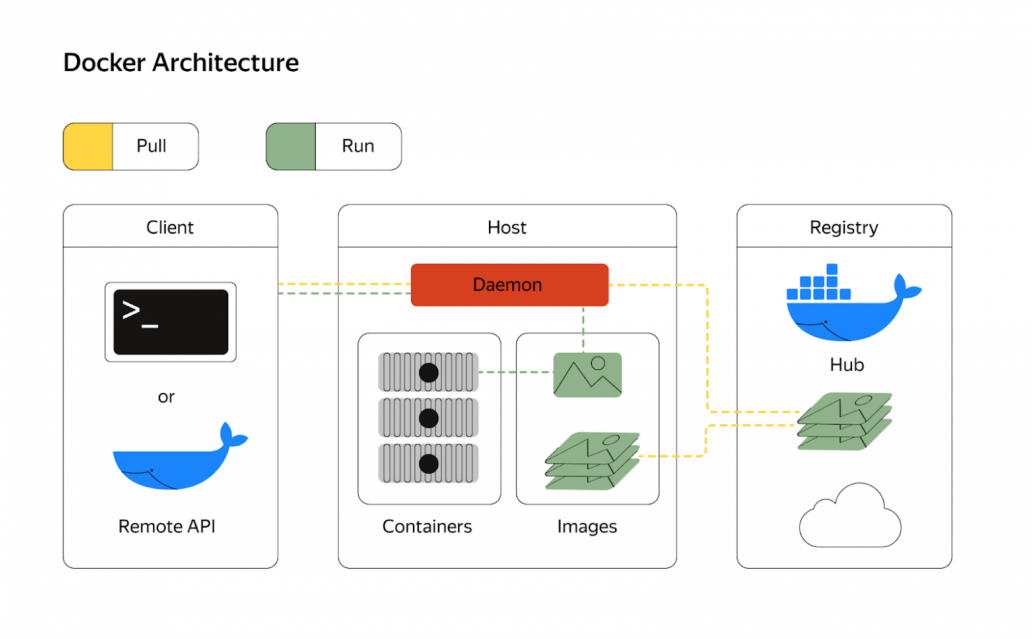

Рисунок 1.3 – Архитектура Docker

1.3.1 Образ

. Образ представляет собой read-only шаблон, который используется для создания контейнеров. Является компонентой сборки Docker. Каждый образ представляет собой набор уровней.

1.3.2 Реестр

Реестр хранит в себе образы. Существуют публичные и приватные реестры. Публичным Docker-реестром является Docker Hub, содержащий большую коллекцию образов, используемых в многих системах. Содержание приватных хранилищ доступно только ограниченном кругу лиц, получившим разрешение на их использование.

1.3.3 Контейнер

В контейнерах содержится вся необходимая для работы приложения информация. Они создаются из Docker-образа. Каждый контейнер изолирован от внешней системы. По сравнению с виртуальными машинами контейнер является более компактным и производительным решением.

1.3.4 Docker Compose

Еще одним элементом Docker, который используется при разворачивании приложений в контейнерах является docker-compose. Данное средство предо-

ставляет удобный способ организации управления несколькими контейнерами, используемыми для совместной работы.

1.4 Http

Http(протокол передачи гипертекста) — клиент-серверный протокол прикладного уровня, используемый для передачи сообщений между клиентами.

1.4.1 Структура протокола

Каждое Http-сообщение состоит из следующих частей:

- стартовая строка, определяющая тип сообщения;
- набор заголовков, которые описывают параметры передачи, тело сообщения и дополнительные сведения о запросе;
- тело сообщения, содержащие собой данные.

Стартовая строка http сообщения указывает тип сообщения и выглядит следующим образом:

```
Метод URI HTTP/Версия # для запроса
HTTP/Версия КодСостояния [Пояснение] # для ответа
```

2 Реализация приложения

2.1 Архитектура системы

В качестве модели архитектуры выберем микросервисную модель.

Для ее реализации использовался язык uml. В состав реализованной в рамках данной работы системы входит три сервиса, взаимодействующих друг с другом по протоколу http и база данных.

Соединение с базой данных происходит только в сервисе `db_handler`. Данный сервис состоит из объектов типа `webapi`, `HandleText`, `HandleQuestion` и `Connector`.

Также в системе находится сервис `scorm_generator`. В данном сервисе реализован тип `Service`, содержащий в себе информацию о генерируемом scorm-пакете.

Чтобы пользователи имели возможность взаимодействовать с системой, в ней располагается сервис `api_handler` предоставляющий пользователям api-интерфейс, с помощью которого они могут обращаться к системе через http-запросы.

2.2 База данных

В качестве базы данных будет использоваться PostgreSQL. Данное решение является одним из популярных, из-за чего существует огромное количество ресурсов, представляющих информацию по работе с этой базой данных.

2.2.1 Модель предоставления данных

Для сохранения данных о каждом тесте реализована реляционная модель представления данных, содержащая всю необходимую информацию о тестовых заданиях и ответах на них.

2.3 Сервис db_connector

2.3.1 пакет Connector

Пакет `Connector` реализован на языке `golang` с помощью пакета `sqlx`. Логика работы с базой данных представлена в методах объекта *DbConnector*.

2.3.2 webapi-интерфейс

`webapi-интерфейс` реализован на языке `golang` с помощью пакета `mux`. Данный пакет содержит необходимые объекты для построения веб-сервисов и

управления ими. Для создания интерфейса были реализованы различные обработчики событий, определяющие поведение сервиса при определенных запросах к нему. Логика работы данного интерфейса описана в реализованном пакете `handler`.

2.4 Сервис scorm_generator

2.4.1 пакет scorm

Пакет `scorm` реализован на языке `golang` с помощью пакета `sqlx`. Логика создания scorm-пакета реализована функцией `GenerateSCORM`.

2.4.2 webapi-интерфейс

`webapi-интерфейс` реализован на языке `golang` с помощью пакета `mux`. Логика работы данного интерфейса в методе `TestSCORMHandler`.

2.5 Сервис app_handler

Для данного сервиса в рамках данной работы были реализованы различные обработчики событий, определяющие поведение сервиса при определенных запросах к нему. Логика работы данного интерфейса описана в реализованном пакете `apphandler`.

2.6 Результат работы

Пример добавления теста в хранилище:

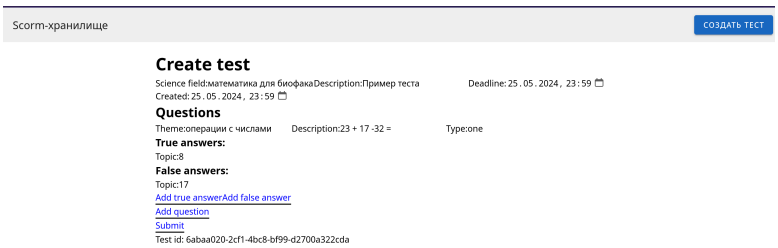

Рисунок 2.1 – Результат добавления теста

На данном изображении мы заполняем простую форму, в которой указываем информацию о тесте, добавляем вопросы и ответы к ним. После нажатия

на `Submit` происходит запрос к реализованной системе и получаем в ответ идентификатор теста.

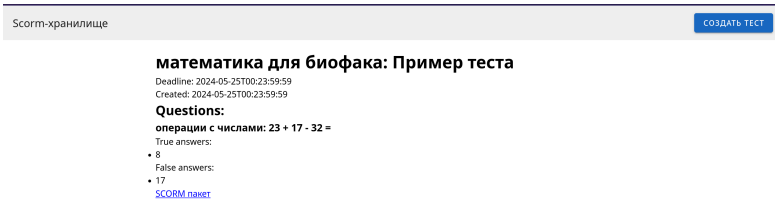

Рисунок 2.2 – Данные теста

Здесь мы запрашиваем информацию о тесте по идентификатору и получаем представление всего содержимого теста. Под этим содержимым есть ссылка для загрузки SCORM пакет, выполняющая загрузку scorm-пакета с разработанной системы. Пример генерируемого scorm-пакета, получаемого из системы:

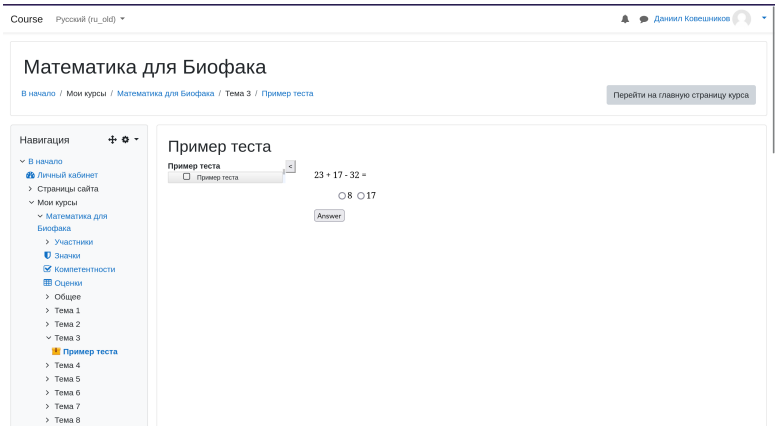

Рисунок 2.3 – Пример генерируемого пакета

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы было произведено ознакомление со структурой scorm-пакетов. Также было проведено ознакомление с различными технологиями для определения их набора, который был использован в данной работе.

В качестве практической части данной работы было выполнено следующее:

- Рассмотрены различные варианты построения архитектуры программного обеспечение и разработана микросервисная архитектура для данного программного обеспечения;
- реализована реляционная модель хранения данных в базе данных postgresql;
- реализован сервис генерации scorm-пакетов, доступ к которому осуществляется через сторонний сервис, предоставляющий пользователю webapi интерфейс;
- реализован механизм доступа к базе данных через sql-speaker объект и webapi интерфейс стороннего сервиса;
- реализован webapi интерфейс для взаимодействия пользователя с программным обеспечением.