МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

СИСТЕМА КОНТРОЛЯ ПОСЕЩЕНИЙ ЗАНЯТИЙ.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ
ПО ДИСЦИПЛИНЕ «Операционные системы и системное программирование»

КП.ПО5.170154 - 04 81 00

Листов 11

Руководитель	Савицкий Ю. В.
Выполнил	Корнасевич И. Д.
Консультант по ЕСПД	Савицкий Ю. В.

Брест 2021

Содержание

Введение. Анализ задачи проектирования **3**

1 Системный анализ и постановка задачи **4**

2 Проектирование системы **5**

2.1 Архитектура БД 5

2.2 Архитектура сервера 6

2.2.1 Repository 8

2.3 Build tools 8

Список литературы **11**

Введение. Анализ задачи проектирования

С появлением многопроцессорных компьютеров параллельное программирование играет важнейшую роль в обработке информации. Сегодня уже невозможно представить процессор только с одним ядром или сервер работающий в однопоточном режиме. Поэтому понимание многопоточности — это полезное умение любого программиста. Большинство операционных систем, особенно интерактивных, работают сразу на всех ядрах процессора одновременно. Также ОС предоставляет интерфейс взаимодействия с потоками и процессами, а также управление их жизненным циклом. В языке C++, как в одном из наиболее низкоуровневых достаточно инструментов работы с потоками:

а) MPI позволяет работать не только в пределах одной машины, но и связывать много процессов в единый кластер.

б) OpenMP предоставляет наиболее простой интерфейс распараллеливания C++ кода, поэтому я предпочитаю использовать его.

в) Thread родной для C++ модуль, но он слегка многословен, поэтому я решил от него отказаться.

На самом деле неважно какие именно инструменты используются при распараллеливании алгоритмов и программ, принципы от этого не меняются.

Задача разработанной программы — обработка изображений. Задача сводится к применению нескольких алгоритмов к матрице байт. Если алгоритм не сложен, то распараллелить его не составит никакого труда.

1 Системный анализ и постановка задачи

Система должна представлять собой клиент-серверное WEB приложение, выполняющее следующие функции:

а) Все пользователи регистрируются в системе администратором.

б) Каждый пользователь имеет роль: преподаватель или студент.

в) Список занятий заносится в базу данных (БД) администратором.

г) Во время очередного занятия, преподавать на своём устройстве начинает занятие, у него появляется одноразовый QR-код, который сканируется студентами.

д) Студент, просканировавший QR-код считается посетившим занятие.

2 Проектирование системы

Система в целом должна состоять из textbfклиента, textbfсервера и textbfБД. Общая схема отображена на рисунке 1.

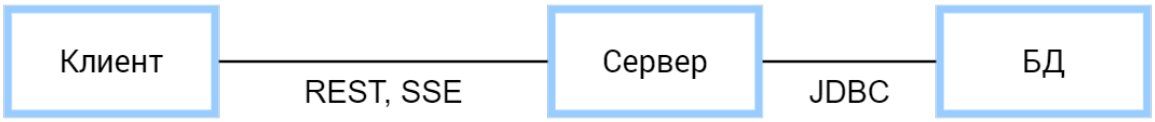

Рисунок 1 – Общая архитектура приложения.

2.1 Архитектура БД

Современные программные системы редко обходятся без энергонезависимого хранилища данных. Это могут быть реляционные, нереляционные (NoSQL), графовые БД. В конце концов данные можно хранить в простых текстовых файлах.

В разрабатываемой системе используется реляционная БД. Это самый простой, хорошо изученный и понятный способ хранить данные. В контексте этого приложения нет смысла использовать NoSQL решения, основные преимущества которых, а именно высокая гибкость, попросту бессмысленны в данной системе.

Как видно на рисунке 2, основные сущности: STUDENT, LESSON, TEACHER, STUDENT_GROUP.

- STUDENT_GROUP позволяет создавать группы студентов, при этом STUDENT и STUDENT_GROUP связаны отношением Many-to-many, что позволяет удобно назначать занятие сразу множеству студентов.
- STUDENT_APPOINTMENT хранит статус посещения занятия студентом.

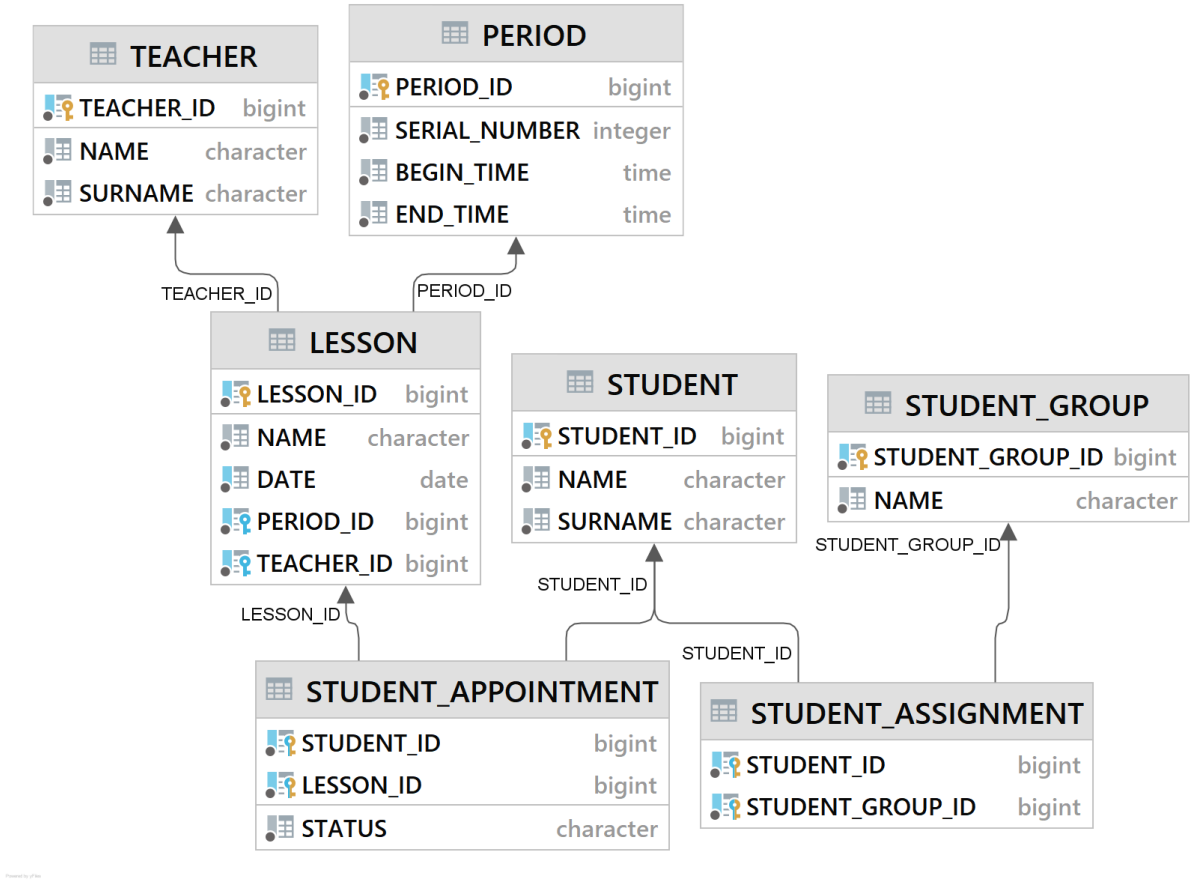

Рисунок 2 – Общая архитектура приложения.

– Каждый LESSON связан с PERIOD. PERIOD здесь выступает номером занятия (1-ая, 2-ая пара и т.д.)

В качестве конкретной БД использована H2, как простое in-memory решение, что сильно ускоряет разработку. В боевом применении лучше использовать H2 в Server или Mixed режиме, также хорошим выбором будет PostgreSQL.

2.2 Архитектура сервера

Серверная сторона должна отвечать следующим критериям:

– Достаточные познания меня, как разработчика, в выбранной платформе.

– Возможность легко взаимодействовать с реляционной БД

– Удобные инструменты реализации REST API

– Поддержка Server Sent Events (SSE)

– Доступные инструменты разработки

– Простота включения внешних библиотек в приложение

К критериям выше идеально подходит Java-Spring [5] платформа. Общая схема сервера на Spring выглядит следующим образом:

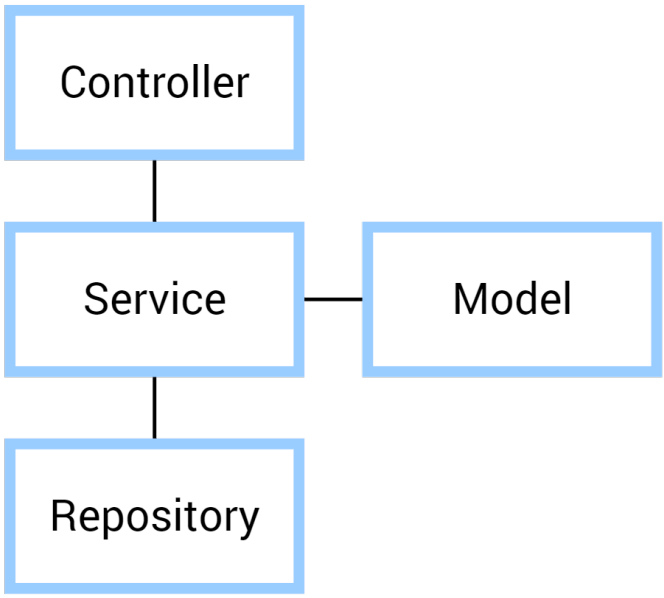

Рисунок 3 – Общая архитектура сервера.

– Controller формирует REST API, также отвечает за создание SSE. При получении HTTP запроса, в этом модуле формируются соответствующие Data Transfer Object (DTO), поля DTO валидируются, после управление передается в Service.

– Service выполняет в первую очередь связующую функцию. Он обращается к Repository и Model по необходимости. Также инфраструктурные задачи лежат в основном на Service.

– Repository отвечает на взаимодействие с БД. В этом слое объекты данные, превращаются в SQL запросы, а результаты этих запросов — в объекты Model.

– В Model хранится бизнес-логика сервера.

2.2.1 Repository

Для взаимодействия с БД в Java обычно используют JPA, в частности Hibernate [2]. В Spring даже есть абстракция над JPA — Spring Data JPA [4]. Но, для реализации поставленной задачи Hibernate, со своей автогенерацией схемы БД, многоуровневым кэшем, дополнительным языком HQL для абстракции над SQL, ленивой подгузкой объектов, также с непонятными SQL-запросами, взявшимися непонятно откуда, кардинально увеличит сложность всего проекта. К тому же я не очень люблю использовать Hibernate из-за того, что он, по факту, заставляет создавать двунаправленный граф объектов, повторяя схему БД, это приводит к раздуванию сервисов, что плачевно сказывается на читаемости кода.

Поэтому, был выбран Spring Data JDBC [3], который к Hibernate и JPA не имеет никакого отношения. Здесь используется Domain Driven Design (DDD) [1], уменьшая размер сервисного слоя, но увеличивая слой доменной модели. DDD предлагает не строить двунаправленный граф объектов, а использовать патерн Aggregate Root. Такой подход позволяет разделить логику на относительно-связанные части — агрегаты. В целом понятность системы растёт, а количество ошибок падает.

2.3 Build tools

Сборка современного Java приложение уже давно перестал быть тривиальной задачей. Причина тому — невероятное количество используемых библиотек. Для облегчения сборки и запуска Java приложений используются специальные инструменты. Самые популярные из них: Apache Ant, Apache Maven, Gradle.

– Apache Ant был одним из первых инструментов автоматической сборки проектов для Java. Ant — императивный инструмент, что каждое действие должно быть описано в файле конфигурации (в данном случае

XML). Это придаёт невероятную гибкость во время сборки, но в тоже время, Ant конфигурация может быть невероятно длинной и сложной.

– Apache Maven в отличие от Ant — полностью декларативный инструмент. Начать новый проект на Maven очень просто, ведь он сам знает, как и куда загружать зависимости проекта, где находится XML конфигурация самого Maven и т.д. От программиста требуется всего-лишь следовать его ожиданиям. Слабая сторона Maven — кастомизация. Для задания дополнительной логики необходимо использовать Maven-плагины, которые не всегда полностью покрывают требования.

– Gradle находит баланс между императивностью Ant и декларативностью Maven. В Gradle отказались от XML заменив его Domain Specific Language (DSL). DSL позволяет декларативно описать сборку на Maven, а если нужно, то написать живой код на Groovy или Kotlin (каждый из которых является JVM языком).

Для проекта выбран Gradle, как одновременно простой, но гибкий инструмент. Gradle также предоставляет разбить цельное приложение на модули.

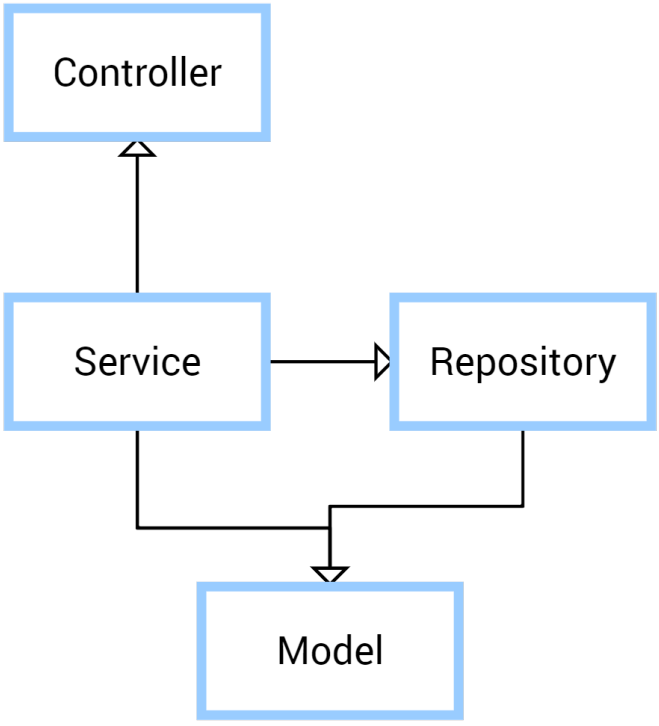

Рисунок 4 – Зависимости модулей друг от друга.

Список литературы

1. *Evans E.* Domain-driven design. — 2003.
2. Hibernate документация [электронный ресурс]. — URL: `https://hibernate.org/orm/documentation/5.6/`.
3. Spring Data JDBC документация [электронный ресурс]. — URL: `https://docs.spring.io/spring-data/jdbc/docs/current/reference/html/#reference`.
4. Spring Data JPA документация [электронный ресурс]. — URL: `https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference`.
5. Spring документация [электронный ресурс]. — URL: `https://docs.spring.io/spring-framework/docs/current/reference/html/`.
6. ГОСТ 19.701-90 ЕСПД [электронный ресурс] : Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. — URL: `https://znaytovar.ru/gost/2/GOST_1970190_ESPD_Sxemy_algori.html`.
7. ГОСТ 7.1-2003 [электронный ресурс] : Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. — URL: `https://www.internet-law.ru/gosts/gost/1560`.