Лабораторная работа 6

Разработка архитектуры приложения

I. Теоретическая часть

Система делится на уровни, каждый из которых взаимодействует лишь с двумя соседними. Поэтому запросы к БД, которая обычно располагается в самом конце цепочки взаимодействия, проходят последовательно сквозь каждый «слой».

Каждый уровень этой архитектуры выполняет строго ограниченный набор функций (которые не повторяются от слоя к слою) и не знает о том, как устроены остальные уровни. Поэтому «содержимое» уровней можно изменять без риска глобальных конфликтов между слоями.

Все сущности оформлены в виде моделей (**Model**).

Они содержат набор атрибутов (приватные поля класса), конструкторы, сеттеры и геттеры для установки/чтения атрибутов. Иного кода в них нет. Часто подобные объекты называют POJO (Plain Old Java Object).

Всю логику работы с моделями реализует слой **Service**.

Он формирует бизнес-правила для моделей. Среди аргументов запросов и возвращаемых результатов часто выступают модели (или их коллекции).

Слой **DAO** — «посредник» между СУБД и Service, работающий непосредственно с базой данных, и отвечающий за взаимодействие с ней.

Каждый слой максимально изолирован от других

Если вместо БД будет набор текстовых файлов, то достаточно поменять только реализацию DAO, не трогая остальной код.

Можно подключить другой Service с минимальными изменениями.

Для описания логики работы с данными, понадобится сервисный слой. Для начала опишем интерфейс с необходимыми методами:

```
public interface StudentService {
  Collection<Student> findAll();
  Student findOneById(int id);
  void create(String firstName, String lastName, int age);
  void update(int id, String firstName, String lastName, int age);
  void delete(int id);
}
```

Метод findOneById в StudentService, если получает от DAO пустой ответ, может возвращать null или пустой экземпляр Optional или даже выбрасывать проверяемое исключение.

В результате у нас получился компонент, реализующий бизнес-логику работы, не имеющий зависимостей от сторонних библиотек.

При реализации сервисного слоя в классе необходимо предусмотреть поле для работы с DAO, который инициализировать в конструкторе:

```
private final Dao studentDao;
public StudentService() {
    this.studentDao = new StudentDao();
}
```

Сам метод получения списка студентов в данном примере не реализует никакой логики обработки и просто запрашивает список от репозитория и возвращает его в слой представления:

```
@Override
public List<Student> findAll() {
     return studentDao.findAll();
}
```

В котроллере добавим закрытое поле для доступа к репозиорию

```
private final StudentService studentService =
                                          new StudentService();
```

В методе контроллера получаем этот список:

```
ArrayList<Student> students = studentService.findAll();
```

Далее с этим списком можно работать в соответствии с логикой программы и, например, отобразить в таблицу на форму.

Аналогично, можно реализовать добавление студента.

В контроллере получаем от пользователя или с формы нужные данные и передаем их в метод сервиса. В сервисном слое эти данные собираем в экземпляр сущности и передаем на сохранение в хранилище (БД):

```
@Override
public void create(String firstName, String lastName, int age) {
     Student student = new Student(firstName, lastName, age);
     studentDao.save(student);
}
```

Задание на лабораторную работу

Разработать и описать архитектуру АИС в соответствии с вариантом.

Выполнить анализ технического задания, описать функциональные требования.

Разработка должна включать в себя:

1. **Слой представления (View)**. Графический интерфейс в виде набора fxml-файлов.
2. **Слой моделей данных (Model)**. Проанализировав задание на разработку, выделить основные сущности. Для каждой из которых создать POJO-классы для представления данных.
3. **Интерфейс сервисного слоя (Service)**. Интерфейс должен включать в себя прототипы методов, реализующих бизнес-логику разрабатываемого приложения.
4. **Интерфейс слоя доступа к данным (DAO)**. Интерфейс должен описывать прототипы методов получения и передачи данных. Данный слой работает с моделями данных, которые записывает в хранилище. Методы должны возвращать не сущности, а Optional<>.

Требования к работе:

1. Описать разработанную архитектуру в виде диаграммы прецедентов, логической и физической моделей данных, диаграммы классов.
2. Разработать структуру пакетов проекта, в которых будут размещаться классы и интерфейсы.
3. Методы сервисных слоев и репозиториев описать в виде руководства пользователя. Описать возвращаемое значение, входные параметры.

Содержание отчета

1. Титульный лист
2. Описание варианта задания
3. Анализ ТЗ, функциональные требования
4. Архитектура проекта: диаграмма классов, логическая и физическая модели данных, диаграмма прецедентов
5. Структура пакетов
6. Скриншоты разработанных форм
7. Руководство программиста к разработанным интерфейсам