МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Технология программирования
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВСКАЯ РАБОТА)

на тему «Разработка программно-аппаратной части модуля маршрутизации для "Online-гид"»

Студент М.Г. Пудовкин _____
(И.О. Фамилия) (личная подпись)

Руководитель к.т.н., доцент, Э. В. Егорова
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

Аннотация

Название бакалаврской работы – «Разработка программно-аппаратной части модуля маршрутизации для "Online-гид"». Объект исследования – туристический сервис Online-гид. Предмет работы – программно-аппаратная часть сервиса.

Цель работы – разработка программно-аппаратной части модуля маршрутизации для приложения «Online-гид». Для достижения данной цели, будет необходимо решить ряд следующих задач:

– Анализ научно-технической литературы при помощи которых будет реализована логика работы сервиса;

– Исследование и анализ работы подобных сервисов для понимания основных характеристик подобных сервисов;

– Разработка программно-аппаратной части сервиса;

– Тестирование разработанного ПО и формулирование заключения по его функциональности.

В первой части рассматривается анализ предметной области, анализ процессов и обобщается необходимость их изменения путем разработки бекэнд составляющей.

Вторая часть описывает процесс разработки программно-аппаратной части модуля маршрутизации.

В третьей части описывается процесс тестирования разработанной части.

Бакалаврская работа выполнена на 45 страницах, состоит из введения, трёх частей, которые включают в себя 29 рисунков, заключения, списка используемой литературы и список используемых источников, включающего в себя 25 источников, из которых 11 на иностранном языке и 1 приложения.

Abstract

The title of the bachelor's work is “Development of the firmware of the routing module for the “Online Guide”.

The object of the study is the online travel guide service.

The subject of the work is the hardware and software part of the service, which will communicate with the client part and the database.

The goal of the work is to develop the firmware of the routing module, which will be further used in the Online Guide project.

To achieve this goal, it will be necessary to solve a number of the following tasks:

– analysis of scientific and technical literature in the field of development of hardware and software;

– research and analysis of the operation of the same services from the point of view of object design to understand the main characteristics of same services;

– development of software and hardware of the service;

– testing the developed software and formulating a conclusion on its functionality.

The first chapter discusses the analysis of the subject area, the basics of the project, analyses the processes and summarizes the need to change them by developing a backend component.

The second chapter describes the process of developing the firmware of the routing module.

The third chapter describes the testing process of the developed part.

The bachelor's work is done on the 45 pages, consists of an introduction, three parts, which include 29 figures, a conclusion, a list of used literature and a list of sources used, including 25 sources, 11 of which are in a foreign language and 1 appendix.

Содержание

Введение

Современные проекты функционируют в условиях большого объема постоянно изменяющейся информации, которую необходимо оперативно анализировать и принимать правильные решения. Большой ошибкой является позиция руководителей проектов, которые, относятся к внедрению информационных систем без должной ответственности. Ведь от этого зависят успешность и прибыль их проекта. Поэтому процесс проектирования информационных систем в настоящее время становиться актуальным.

Актуальность выбранной темы заключается в том, что сервис, как и любой другой программный продукт, активно нуждается в программно-аппаратной части.

Объектом работы является туристический сервис Online-гид.

Предметом данной работы является программно-аппаратная часть сервиса, которая будет осуществлять связь клиентской части и БД.

Целью работы будет разработка программно-аппаратной части модуля маршрутизации, которая будет в дальнейшем использоваться в проекте Online-гид.

Для достижения данной цели, будет необходимо решить ряд следующих **задач**:

– анализ научно-технической литературы в области разработки аппаратно-программных средств, при помощи которых будет реализована логика работы сервиса;

– исследование и анализ работы подобных сервисов с точки зрения объектного проектирования для понимания основных характеристик подобных сервисов;

– разработка программно-аппаратной части сервиса;

– тестирование разработанного ПО и формулирование заключения по его функциональности.

Бакалаврская работа состоит из введения, трёх частей и заключения.

В первой части рассматривается анализ предметной области, основы работы проекта, анализ процессов и обобщается необходимость их изменения путем разработки бекэнд составляющей.

Вторая часть описывает процесс разработки программно-аппаратной части модуля маршрутизации.

В третьей часть описывается процесс тестирования разработанной части.

В заключении представлены основные выводы, сделанные в процессе проведенного исследования, описаны результаты практической реализации.

Бакалаврская работа выполнена на 45 страницах, состоит из введения, трёх частей, которые включают в себя 29 рисунков, заключения, списка используемой литературы и список используемых источников, включающего в себя 25 источников, из которых 11 на иностранном языке и 1 приложения.

1 Анализ предметной области приложения «Online-гид»

1.1 Обзор концептуальной модели приложения «Online-гид»

Перед тем, как перейти к анализу и описанию предметной области, в которой существует проект, стоит дать его краткую характеристику.

Данный проект в конечном виде, к которому стремятся члены команды, представляет из себя программу-гид, которая облегчит организацию культурной программы туристам, проложит маршруты к достопримечательностям.

Приложение имеет интерактивную модель города, где пользователь сможет взаимодействовать с объектами города:

– прочитать о достопримечательностях,

– узнать много нового о них и о городе вообще,

– увидеть многие объекты (на основе применения элементов дополненной реальности).

Теперь, когда имеется общее представление о проекте, можно перейти к анализу предметной области.

Поскольку было необходимо наладить связь клиентской части и БД для работы модуля маршрутизации, то далее анализ предметной области будет осуществляться относительно этой позиции.

Концептуальная модель – это систематизированное содержательное описание моделируемой системы (или проблемной ситуации) на неформальном языке. Неформализованное описание разрабатываемой имитационной модели включает определение основных элементов моделируемой системы, их характеристики и взаимодействие между элементами на собственном языке. При этом могут использоваться таблицы, графики, диаграммы и т.д. Неформализованное описание модели необходимо как самим разработчикам (при проверке адекватности модели, ее модификации и т.д.), так и для взаимопонимания со специалистами других профилей. Концептуальная модель содержит исходную информацию для

системного аналитика, выполняющего формализацию системы и использующего для этого определенную методологию и технологию, т.е. на основе неформализованного описания осуществляется разработка более строгого и подробного формализованного описания [24].

На этапе построения концептуальной модели создается целостное и системное описание используемых знаний, отражающее сущность функционирования проблемной области. От качества построения концептуальной модели проблемной области во многом зависит, насколько часто в дальнейшем по мере развития проекта будет выполняться перепроектирование базы знаний.

Результат концептуализации проблемной области обычно фиксируется в виде графических схем на объектном, функциональном и поведенческом уровнях моделирования:

– IDEF0,

– DFD,

– ER – модель,

– диаграмма классов.

Первые две модели описывают статические аспекты функционирования проблемной области, а третья модель - динамику изменения ее состояний.

Естественно, что для различных классов задач могут требоваться разные виды моделей, а, следовательно, и ориентированные на них методы представления знаний. Рассмотрим каждую из представленных видов моделей.

IDEF0 (Integrated Definition Function Modeling) – методология функционального моделирования. В основе IDEF0 методологии лежит понятие блока, который отображает некоторую бизнес-функцию. Каждая из четырёх сторон блока выполняет свою роль. Слева расположены входные параметры, справа – выходные данные.

В верхней части указаны управляющие элементы, а в нижней – элементы механизмы.

Далее на рисунке 1.1 показана диаграмма, построенная по описываемой методологии. Схема описывает работу объекта исследования – системы выбора оптимального туристического маршрута.

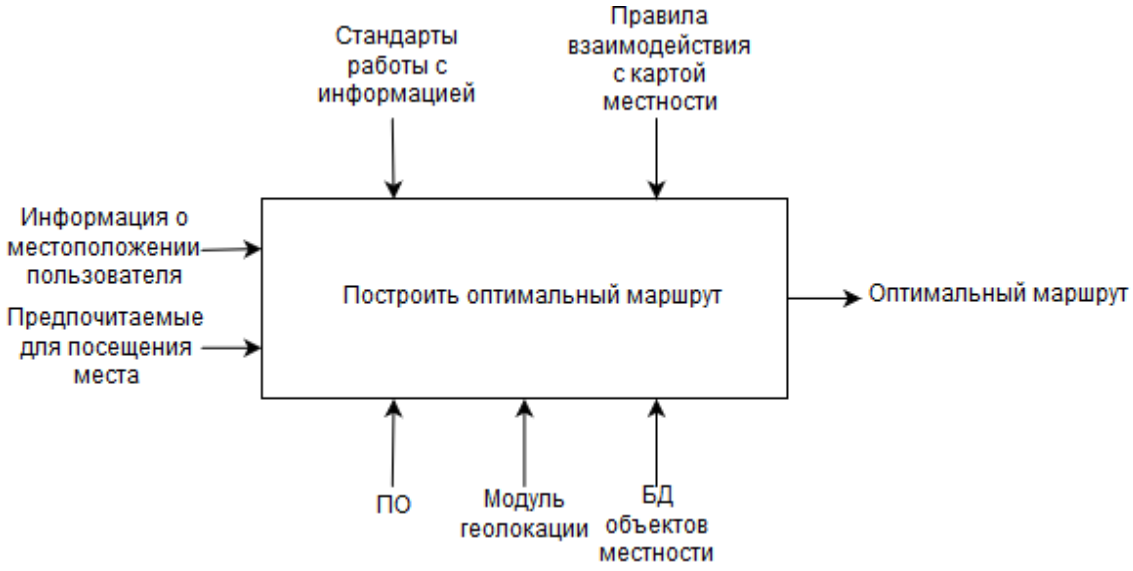

Рисунок 1.1 – IDEF0 диаграмма

Взаимодействие между функциями в IDEF0 представляется в виде дуги, которая отображает поток данных или материалов, поступающий с выхода одной функции на вход другой. В зависимости от того, с какой стороной блока связан поток, его называют соответственно "входным", "выходным", "управляющим".

Одну большую функцию можно разбить на более простые элементы. Этот процесс называется декомпозицией. В данной работе были выделены следующие дочерние функции:

– определить местоположение пользователя,

– предложить доступные места для посещений,

– принять и обработать выбранные пользователем объекты,

– сформировать оптимальный маршрут.

В качестве входного материала каждая нижестоящая подфункция получает выходной материал вышестоящей функции. Также для каждой

дочерней функции были установлены свои элементы управления и их механизмы.

На рисунке 1.2 представлена разработанная IDEF0 декомпозиция процесса работы модуля.

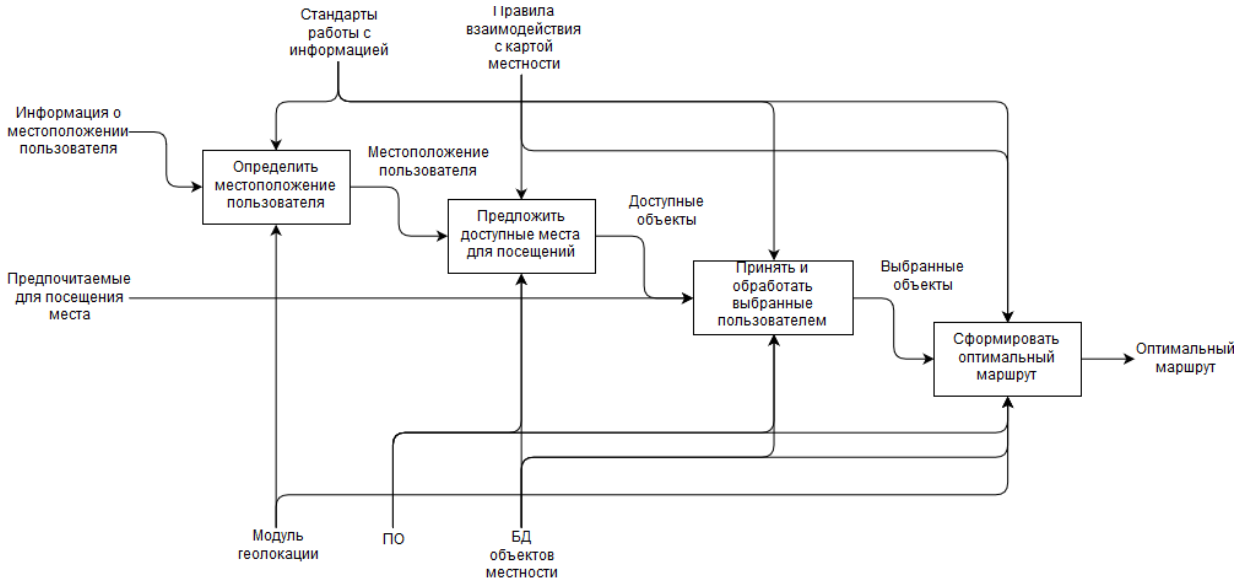

Рисунок 1.2 – IDEF0 декомпозиция основного процесса

В большинстве случаев декомпозиция какого-то обобщённого процесса облегчает понимание работы продукта в целом.

Поэтому в дополнение к обычной IDEF0-модели, описывающей общий процесс, всегда следует прикладывать декомпозицию, представленную разбитием чего-то большого на дочерние подфункции.

1.2 Движение потоков данных приложения «Online-гид»

Диаграмма потоков данных (Data Flow Diagrams — DFD) представляет собой иерархию функциональных процессов, связанных потоками данных. Цель такого представления — продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

Данная методология реализуется в двух вариантах. Первый вариант – Йордана-Коада, и второй – Гейна-Сарсона.

С помощью нотации DFD можно описывать любые процессы, поскольку почти в любой системе происходит некая работа с данными, которую можно представить схематически. DFD удобны при анализе системы в плане оборота документов, поскольку наглядно отображается, где хранятся данные и каким образом производится обмен ими. Легко заметить такую проблему, как дублирование данных или же наоборот – их недостаток в каком-либо месте.

Эта методология никак не учитывает такой параметр, как время. Она лишь показывает, из каких источников появляются те или иные данные, какие данные требуются сущностям и как они обрабатываются этими сущностями. Также в DFD отсутствуют такие элементы, как условия и развилки, которые имеются в той же диаграмме деятельности. Данная нотация нацелена не столько на описание выполнения самого процесса, сколько на работу с данными.

Для отображения процесса работы данных в системе выбора оптимального туристического маршрута также была использована методология DFD. Поскольку это поможет понять, из чего будет состоять система и что нужно для автоматизации данного процесса.

На рисунке 1.3 показано, что во всей системе работает всего две сущности. Сущность «Пользователь» отдаёт данные о местах, которые он хотел бы посетить. В свою очередь сущность построения маршрута, получая данные от пользователя, возвращает данные, в которых указан оптимальный маршрут для посещения объектов.

Рисунок 1.3 – Диаграмма потоков данных

Для нотации DFD также доступна декомпозиция, целью которой, как уже было описано раньше, является упрощение понимания общего процесса. Но в данном случае декомпозиция не была произведена, поскольку данный процесс и без того прост, и его понимание происходит без труда.

1.3 ER-модель приложения «Online-гид»

ER-модель (entity-relationship model) – также называемая модель «сущность-связь», методология позволяющая описывать концептуальные схемы предметной области.

Данная модель используется при проектировании баз данных на высоком уровне. Также с помощью данной нотации можно выделить установить ключевые сущности и связи между ними. Процесс преобразования ER-модели в схему базы данных происходит при проектировании баз данных.

Такая диаграмма не отличается своей детализацией, поскольку в неё включены лишь основные сущности и связи, установленные между ними. Пункт описания первичных ключей в диаграмме сущность-связь можно опустить. Возможно наличие связи «многие-ко-многим».

Но, несмотря на эту простоту, такие диаграммы часто используются для представления структуры данных при обсуждении её с экспертами предметных областей.

Для построения информационной системы в данной работе было решено построить и ER-диаграмму. Но перед тем, как переходить к рассмотрению построенной диаграммы, необходимо ознакомиться с основными понятиями этой нотации.

Сущность – вид схожих объектов, в которых хранится учитываемая информация. Эта информация называется атрибутами сущности. Атрибут хранит в себе свойство объекта. Также они делятся на два типа: собственные и наследуемые. Собственными атрибутами называются те, которые в рамках конкретной модели являются уникальными. Наследуемые атрибуты сущность получает от родительской сущности при установке соответствующей связи. Каждая такая сущность должна именоваться существительным в единственном числе.

Первичный ключ (Primary Key, PK) – значение, которое является уникальным для каждого экземпляра сущности. При создании сущности нужно определить те атрибуты, которые будут являться первичным ключом, и определять каждый экземпляр сущности. Также не стоит забывать, что ни один из атрибутов являющимися первичным ключом не могут иметь нулевое значение. И, конечно же, нельзя менять первичный ключ у экземпляра, поскольку это будет совсем другой экземпляр сущности.

Внешний ключ (Foreign Key, FK) – при определённой связи между двумя сущностями атрибуты одной сущности, являющиеся первичными ключами, будут наследованы второй сущностью в качестве внешних ключей.

Теперь, когда основные понятия построения ER-диаграммы рассмотрены, можно перейти к непосредственному рассмотрению диаграммы, построенной для объекта исследования работы и изображённой на рисунке 1.4.

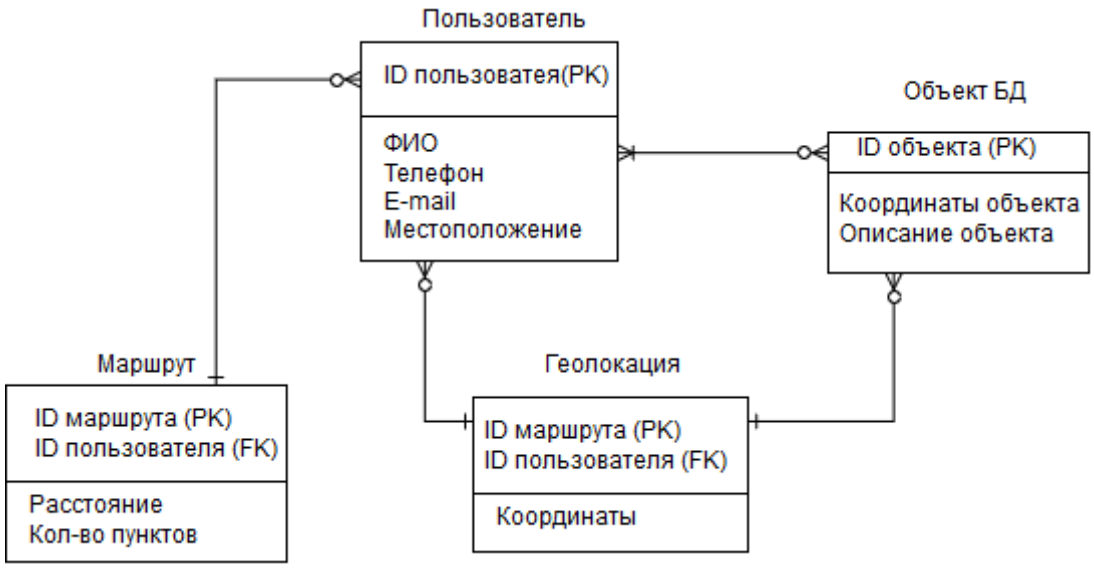

Рисунок 1.4 – ER-диаграмма

На диаграмме присутствует четыре сущности, которые были названы по объектам, которые взаимодействуют в рассмотренной информационной системе. В поля дополнительных атрибутов были добавлены поля, в которых будет записываться информация для каждой сущности. Для каждой были заданы первичные ключи, которые будут идентифицировать каждый экземпляр сущности. Также, например, у сущности маршрута присутствует внешний ключ, который был получен из пользователя. Внешний ключ показывает то, что конкретный маршрут был выдан конкретному пользователю.

1.4 Диаграмма классов приложения «Online-гид»

Одним из важнейших аспектов объектно-ориентированного анализа и проектирования является отображение информационной системы в виде диаграммы классов. Благодаря средствам методологии UML в диаграмме классов можно наглядно и просто изобразить множество информации, которая хранится в системе [17].

С помощью диаграммы классов легко отображается статистическая структура модели информационной системы. Поскольку на прошлых курсах было произведено ознакомление с дисциплиной объектно-ориентированного

программирования, то работа с данной терминологией классов не будет тяжкой. Ведь любую систему легко интерпретировать в виде отношений классов. Объекты конкретной предметной области можно выделить сущности и описать их внутреннюю структуру и типы отношений. Данная диаграмма, также как и две предыдущие, при описании функционировании системы опускает такой аспект как время.

Диаграмма классов может насчитывать большое количество элементов системы, которые представлены в очевидных понятиях языка UML. Эти элементы представлены в виде классов, интерфейсов. И, конечно же, между этими объектами устанавливаются связи. Поскольку эта нотация придерживается стандартов объектно-ориентированного анализа и проектирования, отдельные компоненты диаграммы можно выделить в отдельную диаграмму для модификации в нечто другое, более самостоятельное [21].

Перед построением диаграммы классов для системы выбора оптимального туристического маршрута были изучены основные понятия нотации. Данные основы будут описаны далее.

Класс – в языке UML выступает в роли большого количества объектов, имеющих схожую структуру, поведение и отношения с объектами других классов. В графическом представлении класс выглядит как прямоугольник, разделённый на две части. В первой части расположены атрибуты или, так называемые, переменные. Во второй части класса перечисляются выполняемые в классе операции или же, если изъясняться на языке программирования, методы. Каждый класс подписан сверху [20].

Также помимо описания внутреннего устройства классов, необходимо описать их внешние отношения с остальной структурой диаграммы. Эти отношения описываются с помощью связей. В языке UML есть четыре базовых вида связей:

– зависимость – данный тип связи обозначает, что некие изменения в родительском классе могут повлиять на работу зависимого, но не наоборот;

– ассоциация – такой тип связи, который указывает, сколько объектов, расположенных с каждого конца соединения, может участвовать в данной связи. Иными словами, ассоциация показывает, что родительская сущность включает в себя несколько экземпляров дочернего класса;

– обобщение – или же наследование, связь вида «общее-частное», показывает связь между классом-родителем и классом-потомком. Такой тип отношения вводится в тех случаях, когда появляется вид какого-нибудь класса. Или же, когда в системе объявляются несколько классов со схожим поведением. В последнем случае общие элементы классов выносятся на вышележащий уровень, формируя родительский класс.

Также в диаграмме классов для каждой переменной и метода необходимо указать параметр видимости [13]. Эти обозначения доступности указываются перед именем участника класса. Поскольку спроектированная система оказалась довольно простой, в диаграмме были реализованы только два параметра видимости данных: public (обозначается знаком «+») и private (знак «–»). Помимо этих двух параметров далее в виде списка будут перечислены остальные:

– # - protected,

– / - derived,

– ~ - package,

Но данные модификаторы доступности не были использованы в системе.

Теперь можно перейти непосредственно к проектированию и последующему рассмотрению диаграммы классов на рисунке 1.5.

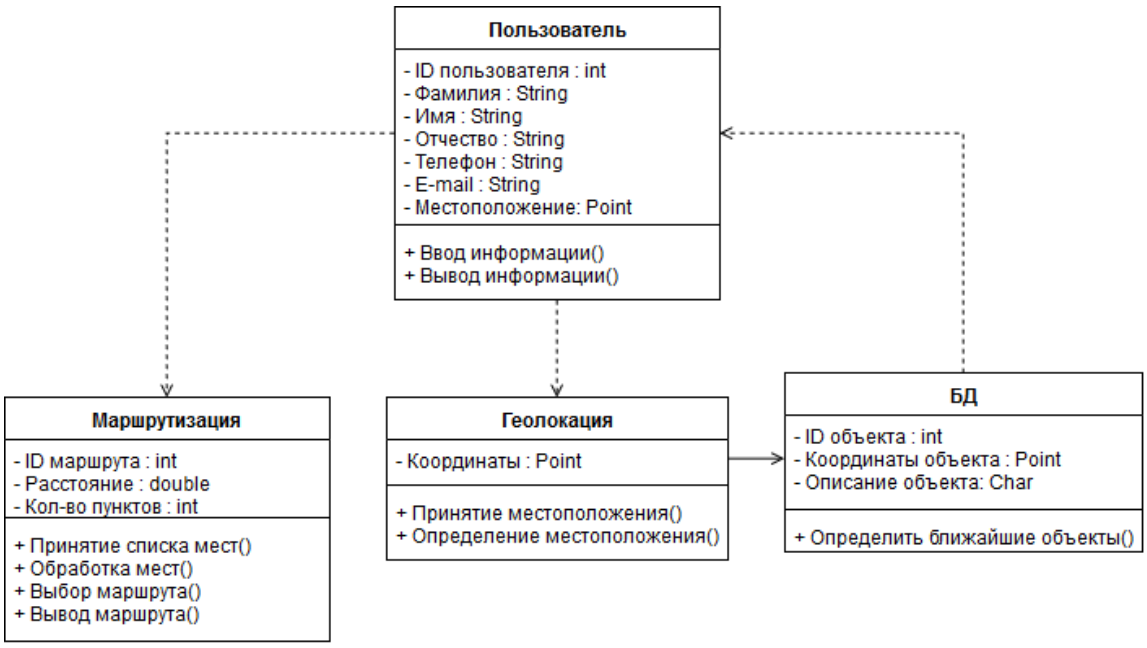

Рисунок 1.5 – Диаграмма классов

Как видно на рисунке 1.5 в данной системе имеется четыре класса. Первый класс «Пользователь» в качестве переменных хранит информацию о пользователе. Каждая переменная обозначена как приватная, поскольку по правилам ООП каждая система должна быть как можно более закрытой. А получение значений переменных из каждого экземпляра будет осуществляться посредством геттеров. Этот класс содержит в себе методы на ввод и вывод информации. Данный класс зависит от класса маршрута.

Остальные классы содержат в себе основную функциональную часть программной части. Как и в первом классе, все переменные обозначены как private, поскольку получение их значений будет осуществляться через геттеры.

На этом описание и проектирование концептуальной модели окончено и далее будет описана логическая модель проектируемой информационной системы.

1.5 Анализ логической структуры приложения «Online-гид»

Логическая модель не только может быть реляционной, иерархической или сетевой, но также показывает логические отношения между атрибутами объектов, не смотря на их содержания и среды хранения. Исходя из всего этого, можно сказать, что логическая модель отображает логические связи между информационными данными в рассмотренной ранее концептуальной модели.

Разные пользователи информационной системы соответствуют различным подмножествам ее логической модели, так называемым внешним пользовательским моделям.

Таким образом, внешняя модель пользователя является отображением концептуальных требований пользователя в логической модели и соответствует идеям, которые пользователь получает о предметной области на основе логической модели.

А это значит, что верность отображения предметной области информационной системы и полнота автоматизированной системы управления данной предметной области напрямую зависит от качества спроектированной внешней модели.

В данном разделе содержится набор диаграмм, моделирующих функциональные возможности и структуру информационной системы на логическом уровне. Исходными данными для диаграмм логической модели служили знания предметной области и диаграммы концептуальной модели информационной системы.

Результатом анализа концептуальной модели информационной системы приложения Online-гид будет изложено в виде графических схем. Эти схемы будут отображать логическую структуру исследуемой предметной области.

Логическая структура будет отображена на диаграммах, которые отобразят функционирующую структуру не только на статических моделях, но и на моделях, демонстрирующих работу в динамике.

В ходе анализа логической модели будут составлены следующие диаграммы:

- диаграмма использования,
- диаграмма деятельности,
- диаграмма последовательности,
- диаграмма компонентов.

По сформированным диаграммам можно будет не только иметь представление о структуре информационной системы приложения «Online-гид». Основываясь на этих диаграммах, можно будет также приступить к разработке программно-аппаратной части модуля маршрутизации.

1.6 Анализ взаимодействий в информационной системе приложения «Online-гид»

Диаграмма вариантов использования (диаграмма использования) – одна из диаграмм языка UML, которая демонстрирует отношения среди актёров и прецедентов. Данная диаграмма описывает то, что будет происходить в системе в процессе её функционирования. Цель данной диаграммы является формулировка общих требований к функциональному поведению разрабатываемой информационной системы [12].

При построении диаграммы вариантов использования проектируемая информационная система представляется в виде множества сущностей или, так называемых, актёров. Эти актёры связаны с прецедентами или же вариантами использования. В качестве актёра может быть кто угодно: человек, техническое устройство, программа или любая иная система, которая может как-то взаимодействовать с объектами проектируемой системы. В свою очередь прецедент является своего рода сервисом, который предоставляется актёру системой. Говоря простым языком, каждый актёр обзаводится каким-то перечнем действий, который он выполняет в ходе работы всей системы.

Далее, на рисунке 1.6 будет приведено рассмотрение спроектированной диаграммы вариантов использования.

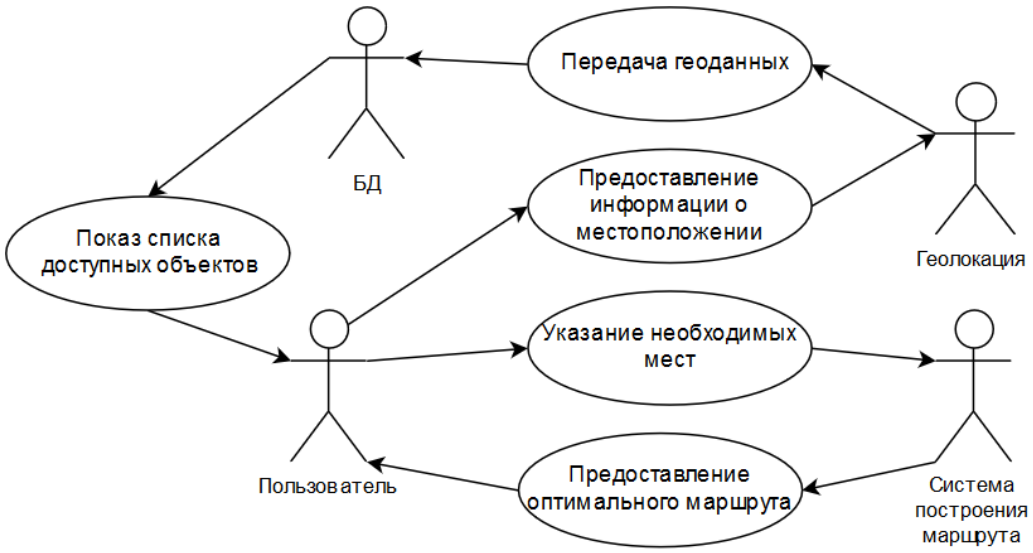

Рисунок 1.6 – Диаграмма прецедентов

В рассмотренной информационной системе преобладают неодушевлённые актёры, поскольку описывается программный продукт. Данная диаграмма наглядно отображает происходящие в системе процессы. Видно, что актёру «Пользователь» доступны четыре прецедента. Это актёр, которому система предоставляет наибольшее количество вариантов использования, поскольку система ориентирована на пользователя. В два из прецедентов он осуществляет непосредственный вклад. Этот актёр является единственной «живой» персоной. Все остальные актёры являются элементами программного обеспечения. Но, тем не менее, тоже вносят значительную лепту в общую работу системы.

1.7 Обзор алгоритма работы приложения с помощью диаграммы последовательности

Диаграмма деятельности – ещё одна из диаграмм, проектируемых с помощью языка UML. Эта диаграмма показывает этапы каких-либо процессов, происходящих в системе. Название диаграммы говорит само за себя. В ней описывается параллельное и последовательное выполнение

элементов. Элементами здесь являются отдельные действия, которые соединены потоками, которые ведут от выходов одних действий к входам других. Данная диаграмма похожа на диаграмму состояний. Диаграмма состояний строится с использованием аналогичных графических элементов языка UML. Различие данных диаграмм заключается в том, что диаграмма состояний отдельные элементы/этапы схемы отображают состояния системы, а в диаграмме действий этапы процессов выражены действиями, выполняемыми во время работы системы. Говоря простым языком, в диаграмме состояний движение идёт по статусным состояниям процесса, а в диаграмме деятельности движение осуществляется через действия процесса.

Перед рассмотрением диаграммы, стоит ознакомиться с основными понятиями построения диаграммы состояний: какие фигуры используются при построении, в каком формате осуществляется построение и т.д. [18].

– закруглённый прямоугольник – в него помещаются действия, которые осуществляются во время функционирования системы;

– ромб – фигура, имеющая несколько выходов, выбор выхода зависит от состояния определённого атрибута, иными словами, данная фигура является состоянием возникающего в ходе работы системы условия;

– широкие полосы – обозначают работу с разветвлёнными действиями, если показано разветвление, то это показывает, что идёт начало параллельного выполнения действия, а если в схеме показано объединение двух параллельно идущих процессов, это свидетельствует об окончании разветвлённого действия;

– чёрный круг – данная фигура обозначает начало процесса и соответственно размещается в самом начале диаграммы;

– чёрный круг с обводкой – обозначает завершение процесса и ставится в конце всей диаграммы.

Проще говоря, данная схема в чем-то схожа со стандартными блок-схемами, строящимися для отображения алгоритма выполнения программы [22].

На рисунке 1.7 можно представлена диаграмма, построенная исходя из анализа системы, которая была выбрана в качестве объекта исследования работы.

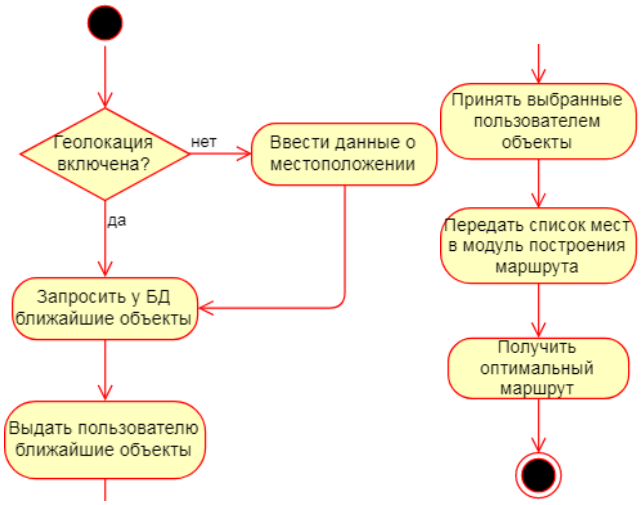

Рисунок 1.7 – Диаграмма деятельности

В размещённой выше диаграмме деятельности показана работа системы. Здесь указаны процессы, состояния которых описано на диаграмме состояний.

1.8 Анализ динамики работы приложения «Online-гид» с помощью диаграммы последовательностей

Диаграмма последовательностей – диаграмма, цель которой, отображение взаимодействий между элементами модели программной системы в терминологии линий жизни и сообщений между ними [14].

Взаимодействие – единица поведения некоторого классификатора, которая фокусируется на наблюдаемом обмене информацией между участниками этого взаимодействия. Графически взаимодействие изображается на диаграмме последовательности как прямоугольник, расположенный на пунктирной линии жизни.

Линия жизни – изображает своего рода течение времени для всех объектов системы. Графически изображается в виде пунктирной вертикальной линии, которая идёт вниз от объекта системы.

Сообщение – связующий элемент, демонстрирующий отношение между двумя элементами схемы.

Комбинированный фрагмент представляет собой мощный механизм моделирования интегрированного поведения, с помощью которого можно представить несколько траекторий взаимодействия в компактной форме для восприятия. В общем случае взаимодействие какой-то конкретной части осуществляется фрагментов взаимодействия.

Комбинированный фрагмент – элемент модели, предназначенный для представления внутренней логической структуры фрагментов взаимодействия. Определяется с помощью оператора взаимодействия и соответствующих ему операндов взаимодействия. Оператор взаимодействия определяет семантику комбинированного фрагмента.

Фрагмент взаимодействия – абстрактное понятие для представления общей единицы или части взаимодействия. Комбинированный фрагмент является частным случаем фрагмента взаимодействия.

Операнд взаимодействия – отдельный фрагмент взаимодействия, предназначенный для использования в качестве внутренней части комбинированного фрагмента. Операнды взаимодействия отделяются друг от друга горизонтальными пунктирными линиями. Операнды взаимодействия образуют фрейм комбинированного фрагмента. Оператор взаимодействия определяет тип комбинированного фрагмента и является перечислением двенадцати литералов: alt, assert, break, critical, ignore, consider, loop, neg, opt, par, seq, strict.

Использование взаимодействия – представляет параметризованную ссылку на взаимодействие в контексте другого взаимодействия. Оно изображается в форме фрейма комбинированного фрагмента с оператором ref, за которым следует полное имя использования взаимодействия.

Теперь, ознакомившись с краткой характеристикой диаграммы последовательности, можно перейти к построению собственной диаграммы на основе сформированной концептуальной модели и знаний предметной области.

Как видно на рисунке 1.8, во время работы системы функционируют 4 элемента.

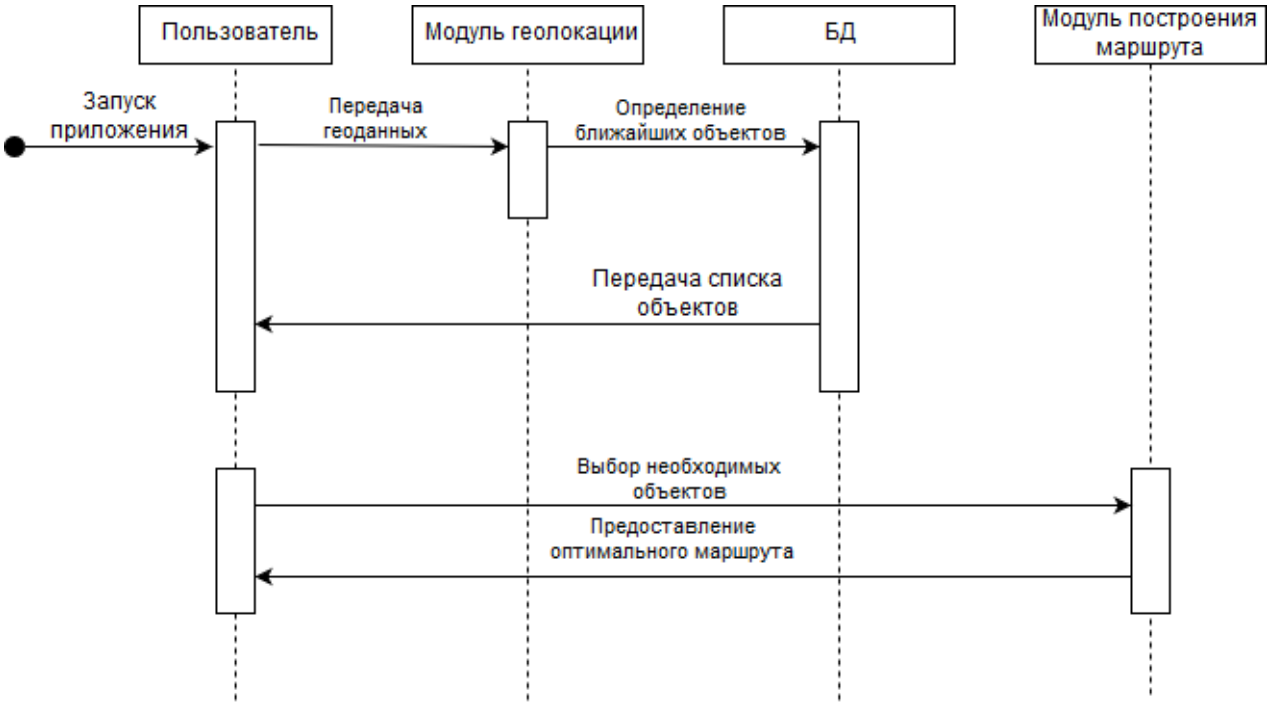

Рисунок 1.8 – Диаграмма последовательности

Сперва пользователь передаёт свои геоданные модулю геолокации, затем БД, получая данные о местоположении, выводит ближайшие места для посещения. Из этого списка, пользователь отправляет модулю построения маршрутов элементы, которые нужно сформировать в какой-то оптимальный путь. Далее работа системы завершается передачей пользователю оптимального маршрута.

1.9 Анализ модулей приложения «Online-гид» с помощью диаграммы компонентов

Диаграмма компонентов строится, чтобы выполнить следующие задачи:

– отображение структуры кода программной системы,

– спецификация исполнимого варианта программной системы,

– добавление свойства рефакторинга для отдельных модулей программной системы.

Проектирование диаграммы компонентов необходимо для обеспечения согласованного перехода представления системы от логического к конкретной реализации проекта в форме программного кода. Все эти компоненты, одни из которых могут существовать только на этапе компиляции программного кода, а другие – на этапе его исполнения, выделяются в отдельные модули для демонстрации модульности разработанного проекта. Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве классификаторов.

Поскольку данная диаграмма называется диаграммой компонентов, имеет место дать определение термину «компонент» в данном контексте.

Компонент – элемент модели, который показывает какой-то модуль системы. Внутри этого компонента обычно помещается некое содержимое, которое определяется самой спроектированной системой. В спецификации языка UML2 компонентом может быть выражен любой автономный элемент системы или подсистемы. Опять-таки не стоит забывать, что каждый компонент, спроектированный в ходе построения одной диаграммы компонентов, может быть повторно использован в другой диаграмме. Это свидетельствует, что в хорошо спроектированной системе, согласно всем стандартам объектно-ориентированного проектирования, каждый компонент самостоятелен и обладает таким свойством, как возможность рефакторинга.

Также каждая часть, что расположена внутри компонента, скрыта и недоступна извне. Но есть элементы, которые предоставляются системой посредством интерфейса. Это ещё один из основных моментов системы, построенной согласно стандартам объектно-ориентированного анализа и проектирования.

Графически компонент выглядит как прямоугольник классификатора с ключевым словом, записанным в форме стереотипа. Внутри символа прямоугольника пишется имя компонента согласно правилам записи имён классификатора. Изображение компонента на диаграмме может содержать также символы интерфейсов, которые присоединяются к прямоугольнику компонента с помощью соединителей. Внутри такого компонента располагаются прямоугольники. Эти прямоугольники – внутренние модули компонента. Каждый такой модуль выполняет свою задачу, необходимую для общего функционирования системы.

Теперь можно перейти к рассмотрению диаграммы компонентов, которая была построена на основе анализа системы выбора оптимального туристического маршрута, на рисунке 1.9.

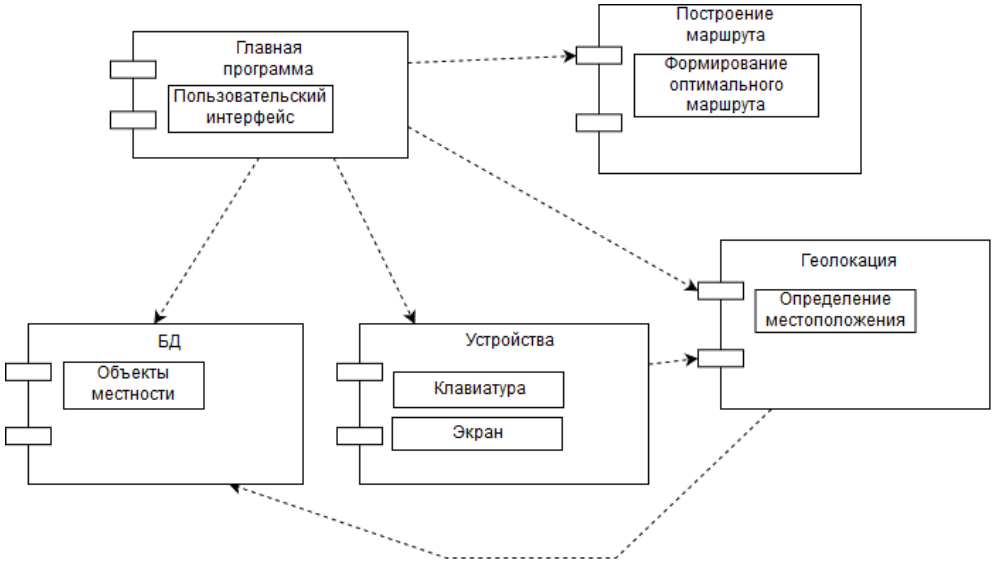

Рисунок 1.9 – Диаграмма компонентов

В системе функционирует пять модулей, потому на диаграмме их такое же количество. В главной программе расположен элемент общения с пользователем – пользовательский интерфейс. Работа с интерфейсом осуществляется с помощью устройств ввода и вывода. Этими устройствами являются клавиатура (ввод) и экран (вывод). Для них выделен отдельный компонент, так как вместе их можно выделить в отдельный отдел, который

управляет работой всей системы, и без управления данная система не имеет смысла.

Выводы по первой части

В первой части был приведён подробный разбор предметной области проекта Online-гид. Была описана концептуальная модель проекта, в которой показаны его функционал и ресурсы. Была описана и логическая модель, цель которой не только то, как объекты системы взаимодействуют между собой, но и показать работу проекта в динамике.

2 Обзор процесса разработки программно-аппаратной части модуля маршрутизации приложения «Online-гид»

Разработка программно-аппаратной части модуля будет состоять из нескольких этапов: создание БД с объектами города, создание DAO (Data Access Object) слоя для работы с ранее сделанной БД, а также реализовать паттерн Repository. Причина, по которой данный паттерн будет реализован, будет описана далее.

Далее будет приведён разбор процесса разработки названых выше частей.

2.1 Создание БД для хранения данных приложения «Online-гид»

Для создания БД использовалась СУБД (система управления базами данных) PosgreSQL. Основными достоинствами этой СУБД являются многоверсионность и высокое качество исходного кода[5].

Теперь перейдём к поэтапному обзору создания БД.

Сперва был осуществлён вход в среду SQL Shell. Здесь будет осуществляться первичная работа с БД. Вход показан на рисунке 2.1.

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (12.2)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
                страницы Windows (1251).
                8-битовые (русские) символы могут отображаться некорректно.
                Подробнее об этом смотрите документацию psql, раздел
                "Notes for Windows users".
Введите "help", чтобы получить справку.
```

Рисунок 2.1 – Вход в Postgress

На рисунке 2.2 показано создание самой БД.

```
postgres=# CREATE DATABASE gid ENCODING 'UTF-8';
CREATE DATABASE
```

Рисунок 2.2 – Создание БД

Как видно из рисунка выше, БД была создана в кодировке UTF-8. Данная кодировка использована из-за своей распространённости и компактности.

Потом, для дальнейшей работы с базой данных, к ней необходимо подключиться. Этот процесс показан на рисунке 2.3.

Рисунок 2.3 – Подключение к БД

Подключившись к БД, необходимо создать таблицы в ней. Создание этих таблиц представлено на рисунке 2.4.

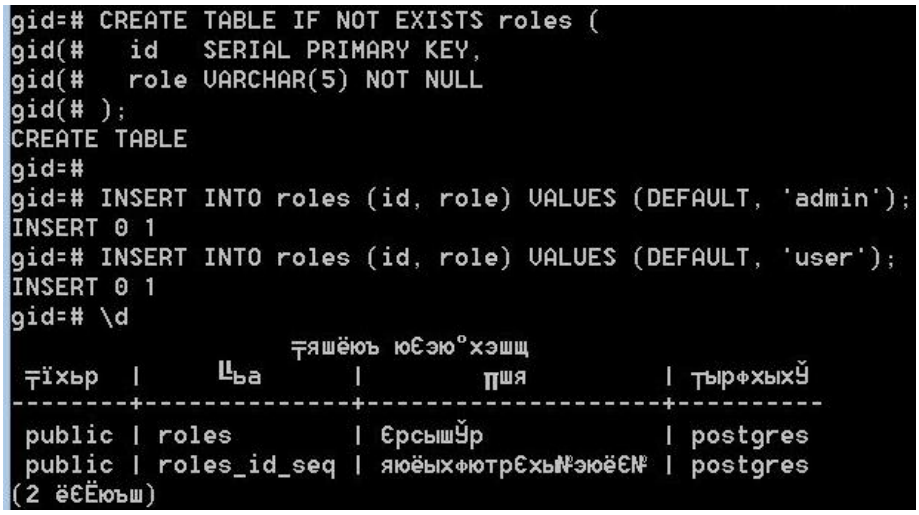

Рисунок 2.4 – Создание таблицы ролей

Из рисунка 2.4 видно, что имеются проблемы с отображением текста. Это вызвано различием кодировок. Об этом было указано ещё при входе PSQL.

Эта проблема решается установкой кодовой странице, которая соответствует Windows кодировке. В дополнение к этому, в свойствах консоли необходимо выбрать любой из доступных шрифтов [16].

Одной из создаваемых таблиц, была таблица пользователей. Её создание показано на рисунке 2.5.

```
gid=# CREATE TABLE IF NOT EXISTS users (
gid(#   id       SERIAL PRIMARY KEY,
gid(#   login    VARCHAR(10) UNIQUE NOT NULL,
gid(#   password VARCHAR(10) UNIQUE NOT NULL,
gid(#   role     INTEGER     NOT NULL,
gid(#   FOREIGN KEY (role) REFERENCES roles (id)
```

Рисунок 2.5 – Создание таблицы пользователей

Далее в таблицу были внесены люди, которые работают с проектом и наделены соответствующими правами.

И наконец, была создана таблица объектов города. Каждый объект города описывался такими полями, как имя, локация, тип объекта, описание и стоимость его посещения. Создание таблицы объектов показано на рисунке 2.6.

```
gid=# CREATE TABLE IF NOT EXISTS city_objects (
gid(#   id   SERIAL PRIMARY KEY,
gid(#   name VARCHAR(30) UNIQUE NOT NULL,
gid(#   location POINT NOT NULL,
gid(#   obj_type VARCHAR(20) NOT NULL,
gid(#   description text,
gid(#   price INTEGER
gid(# );
CREATE TABLE
```

Рисунок 2.6 – Создание таблицы объектов города

После создания таблиц, структура БД отображена на рисунке 2.7.

```
gid=# \d
                  Список отношений
 Схема  |         Имя          |        Тип         | Владелец
--------+----------------------+--------------------+----------
 public | city_objects         | таблица            | postgres
 public | city_objects_id_seq  | последовательность | postgres
 public | roles                | таблица            | postgres
 public | roles_id_seq         | последовательность | postgres
 public | users                | таблица            | postgres
 public | users_id_seq         | последовательность | postgres
(6 строк)
```

Рисунок 2.7 – Структура БД

Также, на данном рисунке видно, как отображается текст в исправленной кодировке.

Для начала, в таблицу с объектами города была добавлена лишь малая часть объектов. Возможность дополнить эту таблицу представится и потом, когда это можно будет сделать через написанное программное обеспечение.

2.2 Добавление возможности работы с данными в БД с помощью паттерна DAO

Data Access Object (DAO) представляет собой прослойку, соединяющую базу данных и систему между собой. Он отвечает за передачу сформированных запросов в базу данных, а также формирует ответы этой БД [10].

Если говорить о DAO в смысле более близком к программированию, то DAO – это интерфейс, который содержит в себе описание методов, с помощью которых будет осуществляться работа между БД и системой. Тогда становиться понятнее, почему данный слой назван так. Всё потому, что он отвечает за доступ к данным. Этот доступ выражен четырьмя операциями: создание, чтение, редактирование и удаление той или иной записи в таблице базы данных [3]-[4].

Далее будет приведено описание разработки DAO-слоя, который будет связывать систему с базой данных.

Первым делом был создан сам интерфейс DAO, в котором были описаны методы для работы с объектами БД. Интерфейс представлен на рисунке 2.8.

```
public interface DAO<Entity, Key> {
    boolean create(Entity model);
    Entity read(Key key);
    boolean update(Entity model);
    boolean delete(Entity model);
}
```

Рисунок 2.8 – Интерфейс DAO

Как и было написано ранее, в интерфейсе описано четыре метода для работы с объектами базы данных: создание, чтение, редактирование и удаление той или иной записи в таблице базы данных.

Из рисунка 2.8 видно, что методы создания, обновления и удаления сущности БД нуждаются в самой сущности. Под эти сущности были написаны два класса под три используемые таблицы.

На рисунке 2.9 показан один из реализующих классов – User.

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {
    private int id;
    private String login;
    private String password;
    private Role role;

    @Override
    public boolean equals(Object o) {...6 lines }

    @Override
    public int hashCode() {...7 lines }

    @Data
    @AllArgsConstructor
    public static class Role {...21 lines }
}
```

Рисунок 2.9 – 2 Класс User под сущности таблицы users

Также стоит обратить внимание на то, что в классе User имеется вложенный класс Role.

Класс Role представляет таблицу roles. В этой таблице содержатся роли, устанавливаемые пользователям.

Это сделано потому, что в БД у каждой записи таблицы users имеется внешний ключ в поле role, который ссылается на таблицу roles.

Таким же образом был сделан класс CityObject, для сущностей таблицы city_objects. Код этого класса представлен на рисунке 2.10.

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
public class CityObject {

    private int id;
    private String name;
    private Double lat;
    private Double lon;
    private String obj_type;
    private String descrirption;
    private Integer price;

    @Override
    public boolean equals(Object o) {...8 lines }

    @Override
    public int hashCode() {...10 lines }
}
```

Рисунок 2.10 – Класс CityObject под сущности таблицы city_objects

На двух, расположенных выше рисунках видно, что был использована библиотека Lombok, которая упростила создание конструкторов и других часто используемых методов.

Однако методы equals и hashCode были переопределены самостоятельно. Это сделано потому, что встроенная аннотация EqualsAndHashCode при переопределении этих методов использует все поля класса. Но две сущности могут быть одинаковыми, имея лишь разный id. Тогда метод equals, переопределённый Lombok'ом посчитает, что сущности различаются, что, по сути, неверно [15].

Теперь, когда сущности под методы интерфейса DAO созданы, можно перейти к реализации в классах

Было реализовано два DAO класса под две таблицы, с которыми будет осуществляться работа.

На рисунке 2.11 показана одна из реализаций интерфейса DAO – класс UserDAO.

```
public class UserDAO implements DAO<User, String> {

    private final Connection connection;

    public UserDAO(final Connection connection) {...3 lines }

    @Override
    public boolean create(User user) {...13 lines }

    @Override
    public User read(String login) {...18 lines }

    @Override
    public boolean update(User user) {...12 lines }

    @Override
    public boolean delete(User user) {...13 lines }

    enum SQLUser {...12 lines }
}
```

Рисунок 2.11 – Класс DAO для работы с таблицей users

В первой строке класса UserDAO видно, что он параметризирован для работы с сущностью User.

Затем была создана вторая реализация, код которой показан на рисунке 2.11.

```
public class CityObjectDAO implements DAO<CityObject, String> {

    private final Connection connection;

    public CityObjectDAO(final Connection connection) {...3 lines }

    @Override
    public boolean create(CityObject cityObject) {...16 lines }

    @Override
    public CityObject read(String name) {...21 lines }

    @Override
    public boolean update(CityObject cityObject) {...12 lines }

    @Override
    public boolean delete(CityObject cityObject) {...17 lines }

    enum SQLCityObject {...12 lines }
}
```

Рисунок 2.11 – Класс DAO для работы с таблицей city_objects

Теперь, когда предоставлены оба рисунка с реализацией методов интерфейса DAO, можно приступить к их обзору.

Сперва описывается, а затем инициализируется в конструкторе класса подключение к базе данных. Когда выполнено подключение к БД, можно приступать к выполнению той или иной операции.

В каждом реализуемом методе выполняется заполнение wildcard'ов, которые выполняют роль подстановочных элементов в SQL запросах [1]. Эти SQL запросы записаны в enum блоке. Заполнив wildcard'ы запроса, его отправляет на исполнение через prepareStatement подключения.

По итогам данной части, был разобран функционал паттерна DAO, благодаря которому осуществляется связь системы и базы данных. Был полностью описан алгоритм его создания. Также была разобрана работа методов этого интерфейса и описано формирование запросов, направляемых в базу данных.

Но в дополнение к этому есть необходимо получения выборки, состоящей из сложных объектов. Для этого был применён паттерн Repository, описание которого приведено далее.

2.3 Реализация паттерна Repository

В дополнение к реализованному паттерну DAO, был также реализован паттерн Repository. Использование ещё одного паттерна было необходимо по следующим причинам:

– во-первых, данный паттерн имеет более обширную область применения, поскольку работает с разными базами данных [19];

– во-вторых, Repository может работать с коллекциями, чтобы возвращать, например, List'ы объектов из БД [9].

В виду этой гибкости, а также возможности работы с данными в сложном формате, было решено применить этот паттерн.

Сперва был создан интерфейс, в котором были описаны методы получения выборок из базы данных. Код интерфейса представлен на рисунке 2.12.

```
public interface SelectionObjectRepository<Entity extends SelectionObject> {

    List<Entity> getSelectionPriceLess(Integer threshold, Comparator<Entity> comp);

    default String wcardTest(final int models) {...15 lines }
}
```

Рисунок 2.12 – Интерфейс Repository

На рисунке выше видно, что метод getSelectionPriceLess возвращает лист неких сущностей. Это не те сущности, что были описаны ранее. Для метода этого паттерна был разработан класс сущности объекта выборки. Код класса показан на рисунке 2.13.

```
@Data
@AllArgsConstructor
@NoArgsConstructor
//@ToString
public class SelectionObject {

    private String objName;
    private Integer price;

    @Override
    public String toString() {...3 lines }
}
```

Рисунок 2.13 – Класс сущности объекта из выборки

Для данного класса был переопределён лишь метод toString. Он был переопределён самостоятельно для корректного отображения [25].

Затем, когда класс для сущности, используемой в описанном методе, сделан, можно перейти к реализации самого метода.

Реализация методов интерфейса Repository представлена на рисунке 2.14.

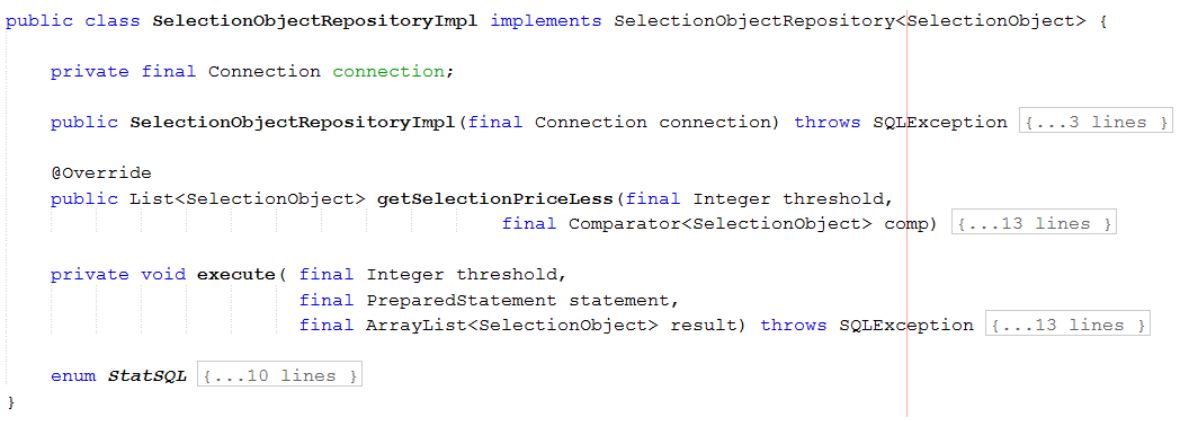

Рисунок 2.14 – Реализация интерфейса

В целом реализация интерфейса Repository похожа на реализацию интерфейса DAO [11]. Описывание и инициализация подключения к БД в конструкторе. После подключения к БД, можно получать выборки из БД.

В реализуемом методе выполняется лишь сам запрос. Заполнение его wildcard'ов происходит в методе execute. Для каждого элемента выборки создаётся дополнительный элемент в Set'е. Этот элемент заполняется полученными в результате запроса данными. Запрос записан в enum блоке [23]. В этот раз он также вынесен в enum блок на случай добавления дополнительных запросов в будущем.

В ходе этой части был разобран паттерн Repository. Был описан алгоритм его создания. Была разобрана работа метода получения выборки и описано формирование запроса, направляемого в базу данных.

Выводы по второй части

Во второй части был описан процесс разработки программно-аппаратной части модуля для проекта Online-гид.

В ходе разработки была создана информационная система, состоящая из БД с объектами города. Для взаимодействия с объектами в базе данных был реализован паттерн DAO. Для получения выборок из базы данных в сложном виде был применён паттерн Repository.

3 Тестирование работы разработанной программно-аппаратной части модуля маршрутизации приложения «Online-гид»

После того, как код был написан, и все паттерны реализованы, необходимо протестировать разработанное. Для этого было решено использовать библиотеку модульного тестирования JUnit. Выбор данной библиотеки обусловлен следующими причинами:

– само тестирование с применением дополнительных библиотек хорошо тем, что исходный код не будет никак изменяться для проверки своей работоспособности [2];

– из-за высокой популярности библиотеки, легко найти техническую документацию;

– легко интегрировать в работу проекта: достаточно прописать зависимость в файл проекта, под тесты создастся отдельный пакет;

– JUnit пишется на том же языке, что и проект – на Java[6]-[8].

Сначала необходимо протестировать работу методов, разработанных при реализации паттерна DAO. Каждый из реализующих интерфейс DAO классов имеет четыре метода. Каждый из этих методов должен быть протестирован.

3.1 Тестирование реализации паттерна DAO

Был разработан модуль с тестами каждого метода класса UserDAO. Сперва было решено поместить подключение к БД в методе setUp, который аннотирован как @Before. То есть перед тем, как выполнить все тесты, выполнялось бы подключение к базе данных.

А затем в методе tearDown, аннотированном, как @After, подключение к базе данных закрывалось бы после выполнения всех тестов.

Но из-за того, что эти методы бы содержали изменение переменных, которые необходимы в методах тестирования, было решено производить

подключение к базе данных и его закрытие в каждом методе тестирования отдельно.

В этом решении есть и положительные моменты. Во-первых, каждый тест будет независимым [7]. И если возникнет необходимость переместить один тест-метод в другой класс. Например, для того чтобы протестировать отдельный метод на различные ошибки и выделить под это отдельный тест-класс. Это сможет быть выполнено, потому что каждый тест имеет своё подключение и его закрытие.

Далее на рисунке 3.1 будет приведён код класса UserDAOTest, в котором реализованы тесты. Также будет проведён анализ каждого теста.

```
public class UserDAOTest {

    public UserDAOTest() {...2 lines }

    @Before
    public void setUp() {...11 lines }

    @After
    public void tearDown() {...7 lines }

    @Test
    public void testCreate() throws SQLException {...10 lines }

    @Test
    public void testRead() throws SQLException {...13 lines }

    @Test
    public void testUpdate() throws SQLException {...12 lines }

    @Test
    public void testDelete() throws SQLException {...15 lines }

}
```

Рисунок 3.1 – Тесты класса UserDAO

Каждый метод в классе UserDAOTest тестирует работу отдельного метода в классе UserDAO. В сумме они тестируют все CRUD операции, осуществляемые над сущностями в базе данных.

Метод testCreate тестирует работу метода create. Создаётся тестовая запись в таблице users. А затем, с помощью функции assertThat, выполняется проверка наличия созданной сущности. Когда проверка выполнена, тестовая запись удаляется, дабы не засорять базу данных.

Метод testRead создан для проверки работы метода чтения записей read. После подключения к базе данных считывается конкретная запись из таблицы пользователей. Далее создаётся сущность типа User, в которую записываются точно такие же данные, что и в считанной ранее записи. В конце с помощью функции assertThat сравниваются: считанная из таблицы users запись и сущность типа User. Если эти две сущности одинаковы, то тест можно считать пройденным, а работа метода testRead законченной.

Потом проверяется работа метода, обновляющего записи в таблице. Конкретно, проверяется способность обновить пароль пользователя. Подключившись к базе данных, метод создаёт сущность пользователя с обновлённым паролем. Потом запись пользователя обновляется согласно новым данным. Далее функция assertThat сравнивает созданную сущность с обновлённой записью, предварительно считав последнюю. Если они совпали, то тест выполнен.

В последнюю очередь тестируется функция удаления пользователей. После того, как подключение к базе данных установлено, создаётся тестовая запись. Эта запись затем считывается и осуществляется подтверждение того, что запись существует. Потом запись удаляется. Дело в том, что метод удаления записи, в случае успеха возвращает значение true. Благодаря этому, в конце работы метода проверяется два bollean показателя. Проверяется то, что запись существовала и то, что она уже удалена и не существует. Если обе проверки возвращают значение true, то тест можно считать выполненным.

В дополнение ко всему написанному стоит заметить, что для тестирования была использована библиотека Hamcrest. А конкретно её функция is. Эта функция помогает синтаксически наглядно показать, что какой-то элемент является одинаковым с тем или иным элементом.

Затем, когда были описаны тесты, проверяющие работа класса UserDAO, необходимо также описать процесс работы тестов для класса CityObjectDAO.

Созданный класс CityObjectDAOTest содержит в себе также четыре метода тестирования для всех CRUD операций взаимодействия с сущностями в базе данных.

Далее будет приведён рисунок 3.2 с разработанным классом, содержащим тесты для класса CityObjectDAO. Также будет кратко описан принцип их работы.

```
public class CityObjectDAOTest {

    public CityObjectDAOTest() {...2 lines }

    @Before
    public void setUp() {...2 lines }

    @After
    public void tearDown() {...2 lines }

    @Test
    public void testCreate() throws SQLException {...12 lines }

    @Test
    public void testRead() throws SQLException {...16 lines }

    @Test
    public void testUpdate() throws SQLException {...12 lines }

    @Test
    public void testDelete() throws SQLException {...15 lines }
}
```

Рисунок 3.2 – Тесты класса CityObjectDAO

Разработанные тесты по своему составу схожи с ранее описанными тестами ввиду того, что оба класса тестируют классы, реализующие один и тот же интерфейс DAO. Потому описание этих методов будет короче. Тем не менее, суть останется ясна.

Метод testCreate создаёт тестовую сущность и записывает её в базу данных. В конце проверяется наличие созданной записи функцией assertThat. Тест считается пройденным, если наличие созданной записи подтверждено.

Метод testRead считывает запись из базы данных. Затем создаётся сущность с такими же данными. Данные из записи сравниваются с данными и сущности. Если они равны, то тест пройден.

Метод testUpdate создаёт обновлённую сущность и записывает её в базу. Затем сравнивается записанная обновлённая запись и сущность с обновлёнными данными. Тест считается пройденным, если они совпадают.

Метод testDelete создаёт тестовую запись и убеждается в её наличии. Затем запись удаляется и проверяется то, что она удалена, возвращаемым методом delete значением true.

3.2 Тестирование реализации паттерна Repository

После того, как классы, разработанные по паттерну DAO, протестированы, необходимо протестировать класс, который реализовал паттерн Repository.

Далее будет приведён рисунок 3.3 с представленным на нём тест-класса. А также будет описан принцип его действия.

```
public class SelectionObjectRepositoryImplTest {

    public SelectionObjectRepositoryImplTest() {...2 lines }

    @Before
    public void setUp() {...2 lines }

    @After
    public void tearDown() {...2 lines }

    @Test
    public void testGetSelectionPriceLess() throws SQLException {...13 lines }

}
```

Рисунок 3.3 – Тесты класса SelectionObjectRepositoryImpl

Разработанный тест-метод проверял работоспособность метода класса SelectionObjectRepositoryImpl, который выстраивал маршрут объектов, стоимость посещения которых ниже заданного значения.

После подключения к базе данных метод инициализирует коллекцию List для сущностей, которые будут точками в сформированном маршруте. В этот лист помещаются те объекты, которые выдаст метод, реализованный по паттерну Repository. Затем каждый объект маршрута подсчитывается и сравнивается с числом объектов, которые действительно входят в маршрут. Если числа совпадают, то тест считается пройденным.

В заключение описания разработанных тестов будут приложены рисунки 3.4 и 3.5, подтверждающие то, что каждый класс прошёл все созданные для него тесты.

```
Running com.mycompany.router.UserDAOTest
read
create
delete
update
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.437 sec

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

Рисунок 3.4 – Результат тестирования UserDAO

Как видно из рисунка, разработанный класс UserDAO прошёл все разработанные тесты, что подтверждает работоспособность его методов.

```
Running com.mycompany.router.CityObjectDAOTest
read
create
delete
update
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.36 sec

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

Рисунок 3.5 – Результат тестирования CityObjectDAO

Класс CityObjectDAO также прошёл все тесты без каких-либо ошибок. Это показано на рисунке 3.6.

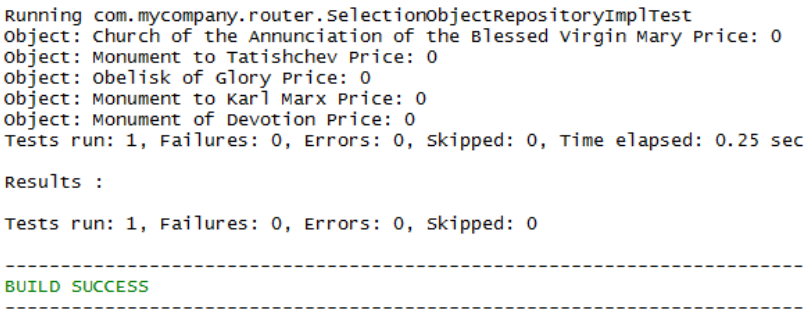

Рисунок 3.6 – Результат тестирования SelectionObjectRepositoryImpl

Класс SelectionObjectRepositoryImpl так же прошёл тестирование.

Исходя из результатов, показанных на трёх рисунках выше, можно сказать, что тесты выполнены успешно, и пора сделать выводы по третьей части в целом.

Вывод по третьей части

В данной части был описан этап тестирования разработанной программно-аппаратной части модуля для проекта «Online-гид».

В ходе тестирования классов, работающих с информационной системой, была рассмотрена библиотека JUnit. С помощью этой библиотеки были разработаны тесты. Разработанные тесты справились со своей задачей и доказали работоспособность программно-аппаратной части модуля проекта. Далее её можно встраивать в бэкенд приложения «Online-гид».

Заключение

Работа, которая была проделана в ходе данной бакалаврской работы, нацелена на создание программно-аппаратной части модуля маршрутизации приложения «Online-гид». Для выполнения этой цели были сформированы задачи, выполнение которых, постепенно приближало ход работы к завершению.

Было проведено исследование работы подобных сервисов с точки зрения объектного проектирования. Это помогло понять основные характеристики подобных сервисов.

После проведённых исследований была проанализирована предметная область приложения «Online-гид». В частности, были рассмотрены концептуальная и логическая модели. Отображение моделей на диаграммах свидетельствует об ещё одном выводе. Понимание структуры процессов, происходящих в системе, помогает лучше понимать и проектировать рабочие модули той или иной информационной системы.

Затем, изучение научно-технической литературы в области разработки аппаратно-программных средств помогло определиться с выбором основных средств, с помощью которых будет создана программного аппаратная часть модуля маршрутизации приложения «Online-гид». А именно, был сделан вывод применить паттерны DAO и Rerository.

Был проведён сравнительный анализ СУБД, в ходе которого было принято решение использовать PostgreSQL. На основе этой СУБД была создана база данных для хранения данных приложения «Online-гид».

В ходе разработки была создана информационная система, состоящая из БД с объектами города. Для взаимодействия с объектами в базе данных был реализован паттерн DAO. Для получения выборок из базы данных в сложном виде был применён паттерн Repository

Был проведён сравнительный анализ средств тестирования. В ходе анализа было решено использовать библиотеку модульного тестирования JUnit.

Были разработаны модули тестирования для методов реализованных паттернов DAO и Repository. Три класса, реализующие интерфейсы были протестированы. Результаты тестов оказались успешными, что говорит о том, что можно было перейти к формулированию заключения по функциональности ПО.

Разработанный модуль можно встраивать в бэкэнд проекта. Он позволяет взаимодействовать с информационной системой проекта Online-гид.

На основании итогов и проделанной работы, можно заявить, что поставленные задачи выполнены. Следовательно, цель данной работы была выполнена.

Список используемой литературы

1. Bloch J. Effective Java Third Edition / Bloch J. – Pearson Education Inc., 2018. – 413p.

2. Cohen F. Java testing and design, from unit testing to automated web tests / Cohen F., Prentice Hall, 2004, 544 p.

3. Core J2EE Patterns - Data Access Object [Электронный ресурс], Режим доступа: https://www.oracle.com/java/technologies/dataaccessobject.html

4. DAO Design Pattern [Электронный ресурс], Режим доступа: https://www.journaldev.com/16813/dao-design-pattern

5. Hans-Jurgen Schonig Mastering PostgreSQL 12, Packt Publishing, 2019 – 470 p.

6. JUnit 5 User Guide [Электронный ресурс]. Режим доступа: https://junit.org/junit5/docs/current/user-guide/index.html

7. Neary D. Unit tests with Junit [Электронный ресурс], Режим доступа: https://web.archive.org/web/20051125204241/http://www.linux.ie/articles/tutorials/junit.php

8. Overview (JUnit 5.6.2 API) [Электронный ресурс]//Технический документ – Режим доступа: https://junit.org/junit5/docs/current/api/

9. Repository Design Pattern [Электронный ресурс]//Статья – Режим доступа: https://medium.com/@krzychukosobudzki/repository-design-pattern-bc490b256006

10. The DAO Pattern in Java [Электронный ресурс], Режим доступа: https://www.baeldung.com/java-dao-pattern

11. The Repository Pattern Explained [Электронный ресурс]//Статья – Режим доступа: http://blog.sapiensworks.com/post/2014/06/02/The-Repository-Pattern-For-Dummies.aspx

12. Буч Г. Введение в UML от создателей языка / Буч Г. - М.: ДМК Пресс, 2017. - 218 с.

13. Вернон В. Реализация методов предметно-ориентированного проектирования / В. Вернон - М.: Вильямс, 2017. – 688 с.

14. Гома Х. UML. Проектирование систем реального времени, параллельных и распределенных приложений / Гома Х. - М.: ДМК Пресс, 2016. - 700 с.

15. Джонсон Р. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Джонсон Р. - М.: Питер, 2016. - 957 с.

16. Документация к PostgreSQL 12.2 [Электронный ресурс]]//Технический документ – Режим доступа: https://postgrespro.ru/docs/postgresql/12/index

17. Ивлев В.А. ABIS. Информационные системы на основе действий / Ивлев В.А., Попова Т.В. - М.: 1С-Паблишинг, 2019. - 248 с.

18. Кватрани Т. Визуальное моделирование с помощью Rational Rose и UML / Кватрани Т. - М.: Вильямс, 2015. - 192 с.

19. Коузен К. Современный Java: Рецепты программирования / Коузен К. – М.: ДМК Пресс, 2018 – 274 с.

20. Ларман К. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку / Ларман К. - М.: Вильямс, 2017. - 736 с.

21. Леоненков А. Самоучитель UML 2 / Леоненков А. - М.: БХВ-Петербург, 2017. - 576 с.

22. Пайлон Д. UML 2 для программистов / Пайлон Д., Питмен Н. - М.: Питер, 2014. - 148 с.

23. Тюгашев А.А. Языки программирования. Учебное пособие / Тюгашев А.А. – М.: Питер, 2016 – 164 с.

24. Фельдман Я.А. Создаем информационные системы / Фельдман Я.А. - М.: Солон-Пресс, 2019. - 432 с.

25. Флэнаган Д. Java. Справочник разработчика / Флэнаган Д., Эванс Б. Дж. – М.: Диалектика-Вильямс, 2019. – 594 с.

Приложение А
Ссылка на исходный код

Исходный код доступен по ссылке https://github.com/pudovkin-m/Router