Министерство образования и науки

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Отчет

по домашней практической работе

по дисциплине **«Проектирование инфокоммуникационных сетей»**

Выполнил: **студент группы**

**K4113с Огирь К.С.**

Проверил: **Осипов Н.А.**

Санкт-Петербург

2020

# Домашняя практическая работа. Разработка программного прототипа по проекту инфокоммуникационной системы

## Цель работы

* Доработать ранее разработанную диаграмму классов;
* Получить практический опыт применения паттернов проектирования GoF.

## Ход работы

Окончательным этапом работы является применение полученных знаний и наработок для программной реализации разработанной диаграммы классов с применением изученных паттернов проектирования GoF. Разработанная диаграмма классов была полезна при выборе паттернов проектирования, всего было выбрано 6 паттернов проектирования:

* Одиночка (Singleton);
* Фасад (Facade);
* Фабричный метод (Factory Method);
* Стратегия (Strategy);
* Строитель (Builder);
* Цепочка обязанностей (Chain of Responsibility).

Часть реализованных паттернов изображены на доработанной диаграмме классов (приложение 1).

Как одно из функциональных требований была описана задача предоставить возможность абитуриенту, формировать электронные заявления в ВУЗы. Т.е. должна быть реализована функция, формирующая электронное заявление в ВУЗ на основе данных системы и данных, предоставленных пользователем. Ни для одного из существующих классов такая функция не являлась характерной, т.е. другие классы решали другие задачи, используя реализованные в них функции. Было принято решение выделить отдельный класс, решающий данную задачу, т.е. реализующий подобную функцию. Далее были рассмотрены варианты, что в будущем при росте системы может помимо ВУЗов в системе могут появиться другие учебные заведения, и в них так-же необходимо было будет отправлять электронные заявления, т.е. для каждой возможной задачи необходимо будет реализовать похожие по реализации классы. Или может произойти так, что необходимо в определенные вузы отправлять немного измененные заявления, в результате была выделена проблема: заранее неизвестны возможные зависимости и типы, с которыми придётся работать, а также необходимость дублировать код для разных версий классов, реализующих данную функцию. Для решения проблемы было принято решение использовать паттерн «Фабричный метод», который также позволит в будущем легко расширять и дополнять систему новым функционалом на основе старого. Для реализации этого паттерна были созданы интерфейсы: Application и ApplicationCreater, а также классы: HEI\_Application и HEI\_ApplicationCreater, реализующие интерфейсы соответственно.

Также в функциональных требованиях указывалось на возможность составления отчетов о приёмной кампании, с аналогией в предыдущем примере данная функциональность была вынесена в отдельный класс. Был выдвинут вопрос: «У каждого ВУЗа есть возможность для составления отчета только о приёмной кампании?». Исходя из логических размышлений было понятно, что: отчет может содержать в себе больше или меньше информации об абитуриентах, ВУЗы могут попросить на основе данных составления большего кол-ва отчетов, чем один, реализация и логика работы классов, составляющих отчеты, отличается лишь некоторым поведением. На основе этой информации было принято решение использовать паттерн «Стратегия», который позволит варьировать поведение некоторых классов во время выполнения программы и позволит вынести отличающееся поведение в отдельную иерархию классов. Для реализации паттерна был создан интерфейс IReport, класс, реализующий его – HEI\_Report и класс, агрегирующий его – ReportBuilder.

Еще одним необходимым для реализации функциональным требованием является возможность фильтрации ВУЗов на основе предпочтений абитуриентов. Для этого за каждым ВУЗом была закреплена коллекция тегов, характеризующих ВУЗ. Но абитуриент может фильтровать ВУЗы не только по тегам, но еще и по городам, в которых они располагаются, или только по городам. Возможно, в будущем добавятся еще группы фильтров. Если осуществлять различные варианты фильтрации через множество перегруженных конструкторов, то при каждом дополнительно введенном фильтре, количество таких конструкторов будет возрастать в экспоненциальной прогрессии. Для решения данной проблемы было решено применить паттерн «Цепочка обязанностей» данный паттерн позволяет обрабатывать различные запросы, такие как фильтрация списка ВУЗов, несколькими обработчиками, количество, и порядок которых можно задавать динамически. Каждый обработчик выполнит сою работу и в результате будет отфильтрованный список определенным образом, в зависимости от запросов абитуриента. Для реализации данного паттерна был создан интерфейс IFilter, абстрактный класс - BaseFilter, реализующий часть функциональности интерфейса, также два класса, наследующих и переопределяющих методы абстрактного класса: FilterByCity и FilterByTags. А также класс HEIsFilterBilder, агрегирующий классы, унаследованные от абстрактного и Задающий начальную точку фильтрации.

Т.к. пре программной реализации классов были использованы средства ASP.NET для построение web API и ORM Entity Framework для работы с базой данных часть паттернов реализована или требует реализации согласно требованиям используемых фреймворков. Так паттерн «Фасад», изолирующий пользователя от ненужной ему реализации системы продиктован Фреймворком ASP.NET. Соответственно было создано два класса, реализующих данный паттерн: AbiturientController и HEIController, разделяющие между собой часть, которой пользуются абитуриенты, и часть, которой пользуются представители ВУЗов.

Также одним из паттернов, которые необходимо было реализовать – это паттерн «Одиночка». Поскольку система работает с базой данных, необходимо чтобы коллекция, содержала актуальные для всех данные. При работе с БД используются коллекции. Запросы напрямую к БД затрачивают много ресурсов, как временных, так и вычислительных, вместе с тем если в системе существует несколько коллекций, содержащих данные из БД, их необходимо постоянно синхронизировать, что также вызывает отрицательное действие на быстродействие системы. Поэтому паттерн «Одиночка необходим» при работе с коллекцией данных, однако предоставляемые используемой ORM EntityFramework возможности уже реализуют данный паттерн, поэтому необходимости в его реализации нет.

Еще одним отмеченным паттерном является паттерн «Строитель» данный позволяющий создавать сложные объекты различных типов. Данный паттерн можно было бы применить для реализации следующей логической структуры: «Объектная модель > Репозиторий (коллекция) > Сервис». Однако данный паттерн был рассмотрен на завершающем этапе разработки, но вместе с тем данный паттерн позволяет создавать подобные скомпонованное объекты подчиняющиеся единым правилам.

## Вывод

В процессе выполнения работы была доработана и уточнена модель диаграммы классов, разработанной ранее, были добавлены интерфейсы, при помощи которых производилась инверсия зависимостей в системе с использованием механизма внедрения зависимостей. Были исправлены отношения между различными классами, указанными неточно или неверно. При доработке диаграммы классов были применены паттерны проектирования GoF такие как: Фабричный метод, Одиночка, Фасад, Стратегия и Цепочка обязанностей, при помощи которых сопровождать и расширят кодовую базу проекта в дальнейшем будет проще. Были также объяснены причины применения выбранных паттернов на для каждого случая. Результатом работы стал разработанный программный прототип системы, соответствующий диаграмме классов. При реализации системы использовались следующие технологии: платформа .NET Core с использованием объектно-ориентированного языка C#, ORM Entity Framework для работой с базой данных как с коллекцией объектов, технология ASP.NET для разработки web API системы для ее тестирования и проверки работоспособности с использованием пользовательского интерфейса Swagger проекта с открытом кодом Swashbuckle.AspNetCore.

## Приложение 1

