# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 4](#_Toc19485458)

[ВВЕДЕНИЕ 5](#_Toc19485459)

[1 ХАРАКТЕРИСТИКА ПРЕДПРИЯТИЯ ООО «ЭНВИЖЕН» 6](#_Toc19485460)

[1.1 Общие сведения о предприятии ООО «ЭНВИЖЕН» 6](#_Toc19485461)

[1.2 Структура предприятия ООО «ЭНВИЖЕН» 7](#_Toc19485462)

[1.3 Техника безопасности на предприятии ООО «ЭНВИЖЕН» 9](#_Toc19485463)

[2 ОПИСАНИЕ ЭТАПОВ ВЫПОЛНЕНИЯ ЗАДАНИЯ 12](#_Toc19485464)

[2.1 Постановка задачи 12](#_Toc19485465)

[2.2 Архитектура приложения 12](#_Toc19485466)

[2.3 Слой доступа к данным (*Data Access Layer*) 14](#_Toc19485467)

[2.4 Слой бизнес-логики (*Business Logic Layer*) 18](#_Toc19485468)

[2.5 Слой представления (*Presentation Layer*) 20](#_Toc19485469)

[ЗАКЛЮЧЕНИЕ 23](#_Toc19485470)

[Список использованных источников 24](#_Toc19485471)

[ПРИЛОЖЕНИЕ А 25](#_Toc19485472)

# ВВЕДЕНИЕ

Производственная практика является важным этапом в подготовке будущих специалистов и проверки полученных ими знаний на практике. Сложно переоценить её ценность в процессе подготовки специалиста.

В ходе технологической практики необходимо решить следующие задачи:

* закрепить на практике знания о подходах разработки приложений;
* получить опыт работы на предприятии;
* получить навыки работы в команде;
* изучить и проанализировать предметную область, технологии и методы, необходимые для разработки приложения;
* разработаны алгоритмы решения поставленной задачи.

Также в ходе прохождения технологической практике необходимо разработать приложение, которое позволило бы вносить данные об энергопотреблении различных предприятий и получать статистику в виде графиков и таблиц, по имеющимся данным. Также при расчёте статистики необходимо учитывать выходные дни, которые могут выпадать на будний день в случае праздников. В связи с этим необходимо также для каждого предприятия хранить список праздников, выпавших на рабочие дни недели.

# 1 ХАРАКТЕРИСТИКА ПРЕДПРИЯТИЯ ООО «ЭНВИЖЕН»

## Общие сведения о предприятии ООО «ЭНВИЖЕН»

На сегодняшний день предприятие ООО «ЭНВИЖЕН» представляет собой компанию, занимающуюся:

* мобильной разработкой;
* разработкой игр;
* разработкой веб-приложений;
* дизайном;
* разработкой программного обеспечения на заказ.

Также данное предприятие – это предприятие, которое первое в Беларуси начало разрабатывать приложения для платформы *iOS*. ООО «ЭНВИЖЕН» предоставляет услуги полного цикла по разработке, внедрению и сопровождению *IT*-решений, имеет успешные результаты работы с компаниями мирового уровня, независимые рейтинги, рекомендации и отзывы авторитетных компаний, подтверждающие их уровень.

Компания ООО «ЭНВИЖЕН» обладает: опытом проектирования и разработки, тестирования и сопровождения программного обеспечения, опытом проведения маркетинговых исследований и рекламных компаний, практическими знаниями в области оптимизации сайтов и продвижения в поисковых системах.

В компании работают высококвалифицированные специалисты, которые помогут быстро и грамотно оформить заказ и ответят на все интересующие вопросы.

Предоставление услуг – специальность данной компании, поэтому в ней собрались десятки талантливых и опытных профессионалов для совместной работы по предоставлению качественных веб-приложений и мобильных приложений. Предприятие ООО «ЭНВИЖЕН» очень заинтересованы в продолжении работы в этой области и готовы решать новые яркие задачи.

Отличительные черты данной компании:

* огромный опыт разработки;
* высокое качество и профессионализм сотрудников;
* ежедневные отчеты о проделанной работе;
* заинтересованность в работе.

Предприятие ООО «ЭНВИЖЕН» было основано в 2009 году.

Правовой статус фирмы ООО «ЭНВИЖЕН» – общество с ограниченной ответственностью. По форме собственности общество имеет частную форму.

Владельцем ООО «ЭНВИЖЕН» является один учредитель Вайгандт Григорий Александрович.

## Структура предприятия ООО «ЭНВИЖЕН»

В компании ООО «ЭНВИЖЕН» функциональный способ организации подразделений (результат развития линейной структуры). Линейный – наиболее простой способ организации. Используется в небольших компаниях или на начальных этапах развития ИТ. Как правило, в этом случае выделяется должность руководителя отдела, в непосредственном подчинении которого находятся несколько ИТ-специалистов.

Схема структуры предприяти представлена на рисунке 1.1.

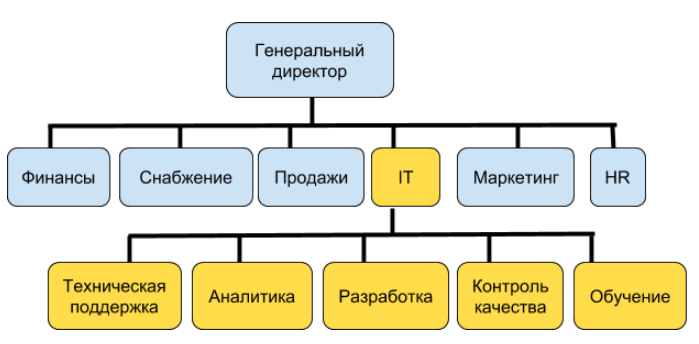


Рисунок 1.1 – Структура предприятия ООО «ЭНВИЖЕН»

Генеральный директор – это руководитель производственно-хозяйственной деятельностью предприятия.

Генеральный директор несет ответственность за сохранность и эффективное использование имущества предприятия, за последствия принимаемых решений, финансово-хозяйственные результаты деятельности.

Функции генерального директора заключаются в общем руководстве производственно-хозяйственной деятельностью компании.

Именно генеральный директор несет полную ответственность за все принятые решения, за результаты деятельности предприятия и сохранность его имущества.

На генерального директора возлагаются следующие должностные обязанности:

* руководство финансовой и хозяйственной деятельностью;
* организация работы с целью осуществления эффективного взаимодействия подразделений;
* организация ведения бухгалтерского учета, обеспечение составления всех форм отчетности;
* утверждение штатного расписания, должностных инструкций для сотрудников;
* принятие мер по обеспечению квалифицированными кадрами;
* принятие мер по обеспечению всем необходимым имуществом;
* обеспечение сохранности имущества предприятия;
* обеспечение защиты интересов в суде;
* осуществление комплекса мер по контролю выполнения сотрудниками требований законодательства РБ, внутренних нормативных документов и должностных обязанностей.

Функции финансового отдела:

* финансовый контроллинг – одна из основных функций финансового отдела, которая заключается в формировании планов и контроля за их исполнением. Исполнение данной функции связанно не только с учетом и анализом, но и контроля за исполнением бизнес-процессов предприятия;
* казначейство. Управление денежными средствами компании, формирование платёжного календаря, контроль за состоянием взаиморасчетов – все это функции казначейства и недооценить их важность невозможно;
* организация и ведение бухгалтерского и налогового учета.

*IT*-директор – это специалист в сфере информационных технологий, который управляет *IT*-подразделением компании и определяет стратегические направления развития технологий для поддержки конкретного бизнеса и является лидером для своих подчиненных в решении технологических задач и достижении поставленных перед подразделением целей. Эта позиция требует системных знаний в информационных технологиях, охватывающих как сферу разработки ПО, так и администрирования и защиты данных. Кроме того, директор в *IT* обладает стратегическим мышлением и опытом управления проектами.

*IT*-директор понимает, как использовать компьютер для менеджмента – он прекрасный управленец хорошо разбирающийся в компьютерах. Также *IT*-директор отвечает за качественное и своевременное исполнение работы подразделения по разработке информационно-программных систем, руководство проектами любой сложности, в том числе комплексными (с участием партнеров и подрядчиков). Проектно-технических решений он не принимает, но способен понимать и оценивать их. Осуществляет в основном управленческо-организаторскую деятельность. Управляет как деятельностью непосредственно подчиненных ему руководителей программ, менеджеров проектов, так и подрядчиков и партнеров.

## Техника безопасности на предприятии ООО «ЭНВИЖЕН»

***1.3.1*** Общие требования техники безопасности при работе на персональном компьютере.

К работе на персональном компьютере допускаются лица, прошедшие обучение безопасным методам труда, вводный инструктаж, первичный инструктаж на рабочем месте.

При эксплуатации персонального компьютера на работника могут оказывать действие следующие опасные и вредные производственные факторы:

* повышенный уровень электромагнитных излучений;
* повышенный уровень статического электричества;
* пониженная ионизация воздуха;
* статические физические перегрузки;
* перенапряжение зрительных анализаторов.

Работник обязан:

* выполнять только ту работу, которая определена его должностной инструкцией;
* содержать в чистоте рабочее место;
* соблюдать режим труда и отдыха в зависимости от продолжительности, вида и категории трудовой деятельности;
* соблюдать меры пожарной безопасности.

Рабочие места с компьютерами должны размещаться таким образом, чтобы расстояние от экрана одного видеомонитора до тыла другого было не менее 2,0 м, а расстояние между боковыми поверхностями видеомониторов – не менее 1,2 м.

Рабочие места с персональными компьютерами по отношению к световым проемам должны располагаться так, чтобы естественный свет падал сбоку, преимущественно слева.

Оконные проемы в помещениях, где используются персональные компьютеры, должны быть оборудованы регулируемыми устройствами типа: жалюзи, занавесей, внешних козырьков.

Рабочая мебель для пользователей компьютерной техникой должна отвечать следующим требованиям:

* высота рабочей поверхности стола должна регулироваться в пределах от 680 до 800 миллиметров; при отсутствии такой возможности высота рабочей поверхности стола должна составлять 725 мм;
* рабочий стол должен иметь пространство для ног высотой не менее 600 мм, глубиной на уровне колен не менее 450 мм и на уровне вытянутых ног не менее 650 мм;
* рабочий стул (кресло) должен быть подъемно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также – расстоянию спинки от переднего края сиденья;
* рабочее место должно быть оборудовано подставкой для ног, имеющей ширину, не менее 300 мм, глубину не менее 400 мм, регулировку по высоте в пределах до 150 мм и по углу наклона опорной поверхности подставки до 20 градусов; поверхность подставки должна быть рифленой и иметь по переднему краю бортик высотой 10 мм;
* рабочее место с персональным компьютером должно быть оснащено легко перемещаемым пюпитром для документов.

Для нормализации аэроионного фактора помещений с компьютерами необходимо использовать устройства автоматического регулирования ионного режима воздушной среды.

Женщины со времени установления беременности и в период кормления грудью к выполнению всех видов работ, связанных с использованием компьютеров, не допускаются.

За невыполнение данной Инструкции виновные привлекаются к ответственности согласно правилам внутреннего трудового распорядка или взысканиям.

***1.3.2*** Требования техники безопасности перед началом работы.

Подготовить рабочее место.

Отрегулировать освещение на рабочем месте, убедиться в отсутствие бликов на экране.

Проверить правильность подключения оборудования к электросети.

Проверить исправность проводов питания и отсутствие оголенных участков проводов.

Убедиться в наличии заземления системного блока, монитора и защитного экрана.

Протереть антистатической салфеткой поверхность экрана монитора и защитного экрана.

Проверить правильность установки стола, стула, подставки для ног, пюпитра, угла наклона экрана, положение клавиатуры, положение «мыши» на специальном коврике, при необходимости произвести регулировку рабочего стола и кресла, а также расположение элементов компьютера в соответствии с требованиями эргономики и в целях исключения неудобных поз и длительных напряжений тела.

***1.3.3*** Требования техники безопасности во время работы.

Работнику при работе на ПК запрещается:

* прикасаться к задней панели системного блока (процессора) при включенном питании;
* переключать разъемы интерфейсных кабелей периферийных устройств при включенном питании;
* допускать попадание влаги на поверхность системного блока (процессора), монитора, рабочую поверхность клавиатуры, дисководов, принтеров и других устройств;
* производить самостоятельное вскрытие и ремонт оборудования;
* работать на компьютере при снятых кожухах;
* отключать оборудование от электросети и выдергивать электровилку, держась за шнур.

Продолжительность непрерывной работы с компьютером без регламентированного перерыва не должна превышать 2-х часов.

Во время регламентированных перерывов с целью снижения нервно-эмоционального напряжения, утомления зрительного анализатора, устранения влияния гиподинамии и гипокинезии, предотвращения развития познотонического утомления выполнять комплексы упражнений.

# 2 ОПИСАНИЕ ЭТАПОВ ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

## 2.1 Постановка задачи

На период прохождения производственной практики на предприятии «Энвижен» была поставлена задача разработать приложение, которое позволило бы вносить данные об энергопотреблении различных предприятий и получать статистику в виде графиков и таблиц, по имеющимся данным. Также при расчёте статистики необходимо учитывать выходные дни, которые могут выпадать на будний день в случае праздников. В связи с этим необходимо также для каждого предприятия хранить список праздников, выпавших на рабочие дни недели. Таким образом приложение должно содержать следующие страницы:

* страница со списком предприятий;
* страница добавления нового предприятия в список;
* страница со списком дат выходных на предприятии, выпавших на будни;
* страница добавления даты выходного на предприятии;
* страница с данными о потреблении энергии на предприятии;
* страница добавления запси о потреблении энергии за указанную дату;
* страница вывода статистики потребления энергии на предприятии за указанный период.

## 2.2 Архитектура приложения

Проблема масштабируемости, повторного использования кода, модулей, автономности модулей поднимается при решении больших задач и проектировании сложных систем и решений [1]. Одним из распространенных подходов решения большинства из этих задач является разделение на слои. Есть один независимый слой, который находится в центре архитектуры. От этого слоя зависит второй, от второго – третий и т. д., общая схема многоуровневой архитектуры изображена на рисунке 2.1. Количество слоев может отличаться, но в центре всегда находятся классы моделей, которые используются в приложении и объекты которых хранятся в базе данных.

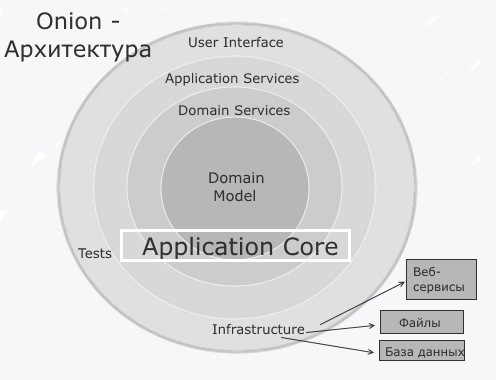


Рисунок 2.1 – Общая схема многоуровневой архитектуры

Слои располагаются друг над другом и содержат логически схожие компоненты с примерно одинаковым уровнем абстракции. Суть в том, что нижележащие слои максимально свободны от вышележащих. Компоненты из одного слоя могут взаимодействовать только с компонентами своего слоя, а также с компонентами более низких слоев.

Разделение на слои делится на гибкое и строгое. Гибкое разделение характеризуется тем, что определенный слой может взаимодействовать со всеми нижележащими слоями. Строгое разделение же подразумевает то, что определенный слой может взаимодействовать только с ближайшим нижележащим слоем. Любое гибкое разделение можно привести к строгому, пробросив взаимодействие через промежуточные слои.

Для решения поставленной задачи было решено использовать классическую трехуровневую архитектуру (рисунок 2.2), подразумевающую разделение приложения на три слоя: слой доступа к данным (*Data Access Layer*), слой бизнес-логики (*Business Logic Layer*), слой представления (*Presentation Layer*).

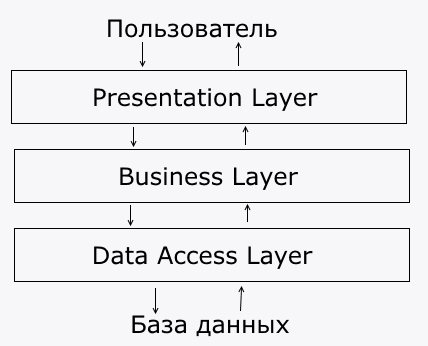


Рисунок 2.2 – Схема классической трехуровневой архитектуры

Слой представления (*Presentation Layer*) – слой, с которым пользователь непосредственно взаимодействует [2]. Этот слой включает в себя механизмы получения команд (ввода) от пользователя, компоненты пользовательского интерфейса (*UI* – *User Interface*), модели представлений, контроллеры, объекты контекста запроса.

Слой бизнес-логики (*Business Logic Layer*) – слой, отвечающий за обработку данных, полученных от слоя представления и реализующий логику приложения. Он взаимодействует с хранилищем данных через слой доступа к данным и передает слою представления результат обработки.

Слой доступа к данным (*Data Access Layer*) – слой, в котором хранятся модели, описываются сущности, размещаются специфичные классы для работы с разными технологиями доступа к данным (например, класс контекста данных *Entity Framework*).

Подробнее о каждом из слоев.

## 2.3 Слой доступа к данным (*Data Access Layer*)

Этот слой содержит все модели данных, хранящихся в базе данных (БД), а также классы, через которые идет взаимодействие с БД. В основе концепции лежит принцип управляемого специализируемого хранения данных, который вводит унифицированный и контролируемый способ доступа к различным данным для приложения [3]. Схема концепции изображена на рисунке 2.3.

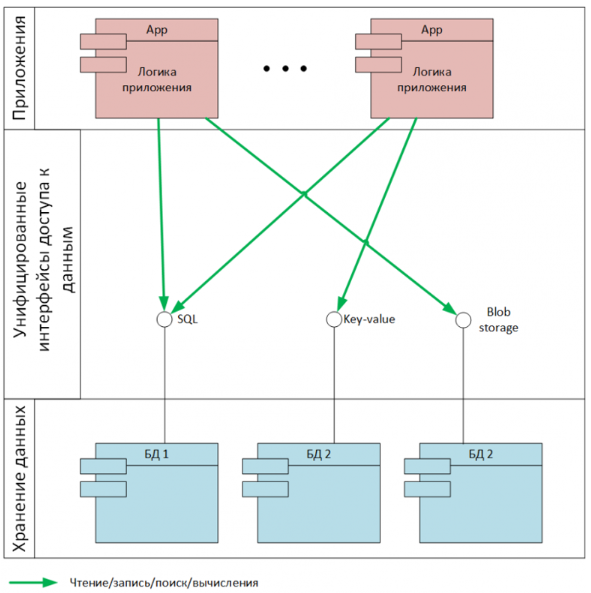


Рисунок 2.3 – Принцип управляемого специализированного хранения данных

В этом слое определяется фиксированный набор классов данных, для которых в инфраструктуре предоставляются некоторые средства хранения. Каждый класс данных при этом предполагает специализированный интерфейс доступа на логическом уровне, оптимальный для данного класса. Например, для класса данных «*key-value*» интерфейс доступа должен предоставлять операции чтения и записи данных по ключу.

Основные паттерны, которые используются в *DAL*-слое:

* «*Repository*»;
* «*Unit of Work*»;
* «*Dispose*».

Для хранения информации о работниках используются три модели, представленные классами «*Consumption*», «*Factory*» и «*Holiday*». Также специально для репозитория типа *Entity* *Framework* используется класс контекста данных *Entity Framework*.

Для увеличения гибкости подключения к хранилищам данных используются репозитории, реализующие один интерфейс репозиториев *IRepository*. Поскольку будет использоваться несколько репозиториев для каждой сущности, то для упрощения работы с сущностями внутри репозитория применён паттерн «*Unit* *of* *Work*». Этот паттерн позволяет упростить работу с различными сущностями в пределах одного репозитория и дает уверенность, что в конкретный момент используются репозитории одного конкретного типа. Он предоставляет доступ к репозиториям через отдельные свойства и определяет общий контекст для всех репозиториев. Для хранения реализаций данных интерфейсов в слое *DAL* определены классы репозиториев «*ConsumptionsEfRepository*», «*FactoriesEfRepository*», «*HolidaysEfRepository*» и класс «*EFUnitOfWork*». «*EF*» в имени репозитория означает репозиторий *Entity* *Framework*. *UML*-диаграмма классов репозиториев «*Factories*», «*Consumptions*» и «*Holidays*» изображена на рисунке 2.4.

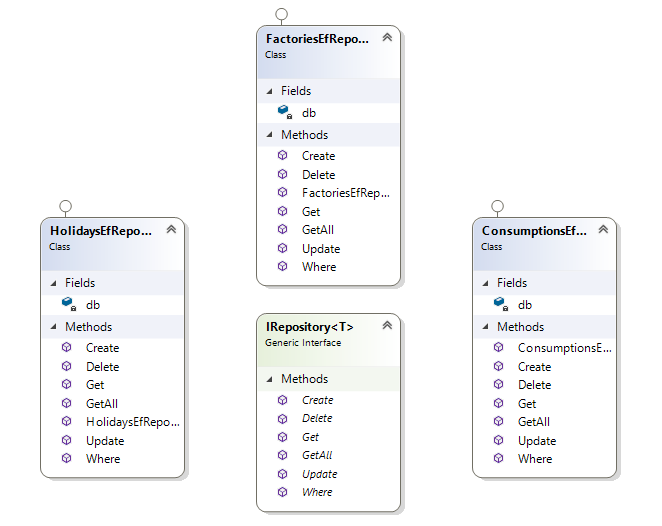


Рисунок 2.4 – *UML*-диаграмма классов репозиториев «*Factories*», «*Consumptions*» и «*Holidays*»

Класс «*EFUnitOfWork*» в конструкторе принимает строку – название подключения, которая потом будет передаваться в конструктор контекста данных. Собственно, через «*EFUnitOfWork*» и будет происходить взаимодействие с репозиторием типа *EF*. В этом классе также содержатся дополнительные методы *Save* и *Dispose*. Метод *Save* служит для сохранения изменений в репозитории, а метод *Dispose* реализует паттерн «*Dispose*». Идея паттерна «*Dispose*» состоит в следующем: вся логика освобождения ресурсов помещается в отдельный метод, который будет вызываться из метода *Dispose*, и из финализатора, при этом добавляя флаг, который говорит о том, откуда был вызван этот метод. На рисунке 2.5 изображена *UML*-диаграмма классов-реализаций интерфейса *IUnitOfWork*.

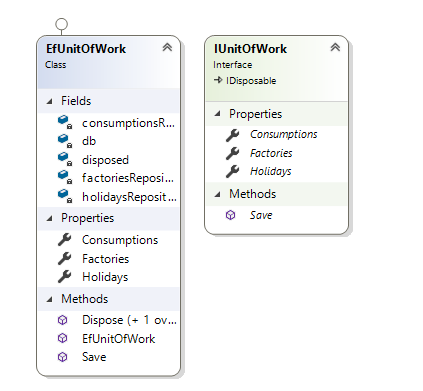


Рисунок 2.5 – *UML*-диаграмма классов-реализации интерфейса *IUnitOfWork*

В итоге приложение не получает прямого доступа к хранилищам данных, а работает через специализированный модуль доступа к данным – *Data Access Layer*. Итоговая структура слоя доступа к данным представлена на рисунке 2.6.

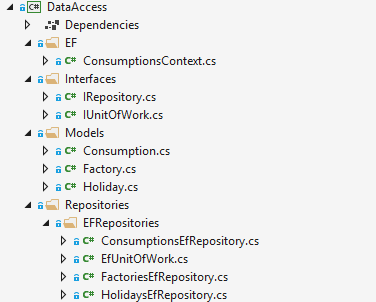


Рисунок 2.6 – Структура *DAL*-слоя

## 2.4 Слой бизнес-логики (*Business Logic Layer*)

Слой бизнес-логики следует на этапе разработки сразу за слоем доступа к данным. Этот слой инкапсулирует всю логику, все необходимые вычисления, получает объекты из уровня доступа к данным и передает их на уровень представления, либо, наоборот, получает данные с уровня представления и передает их на уровень данных [4]. В нем используются классы из слоя доступа к данным.

Основной паттерн, который используется в *BLL*-слое – «Абстрактная фабрика» – интерфейс для создания сервисов работы с *DAL*-слоем.

Для того, чтобы начать использовать *BLL*-слой в проекте, необходимо добавить ссылку на *DAL*-слой, т.к. тут будут использоваться классы, определенные в *DAL*-слое. Слой представления не может получать данные из хранилища данных напрямую. Поэтому *BLL* будет выступать посредником между этими двумя слоями. Но слой *BLL* не может также передавать напрямую в контроллеры объекты своих классов, т.к. слой представления не должен иметь доступ к функциональности слоя *DAL*. Для этого в слое *BLL* используются классы «*FactoryDTO*», «*ConsumptionDTO*» и «*HolidayDTO*». Через эти классы передаются объекты между уровнями. Хотя данные классы во многом похожи по определению на классы «*Factory*», «*Consumption*» и «*Holiday*», это необязательное условие. Классы *DTO* должны содержать только те данные, которые необходимо передавать на уровень представления или, наоборот, получать с этого уровня. То есть это то, что называется *Data* *Transfer* *Object* – специальная модель для передачи данных.

Для упрощения сопоставления классов моделей в проекте используется библиотека *AutoMapper*. *AutoMapper* – шаблон, позволяющий проецировать одну модель на другую. Он проверяет, есть ли соответствующие поля в указанных типах, соответствие проводится как по имени свойства, так и по его типу. Как это работает: вначале создается объект репозитория *IRepository*. Затем в каждом методе контроллера производится конфигурация *Automapper* и сопоставление объектов.

Для настройки конфигурации используется метод *Mapper.Initialize*(). В этот метод передается параметр конфигурации, который с помощью различных методов, в частности, метода *CreateMap*, настраивает сопоставление классов. При конфигурации сопоставляются одноименные свойства классов сущности и модели отображения. При сопоставлении объектов используется метод *Mapper.Map*, причем так же можно создавать коллекции объектов.

Большую роль в приложении играет валидация данных [6]. По большей части за валидацию отвечает именно *BLL*. В контроллере не возникает проблем провалидировать модель через объект *ModelState* и при необходимости возвратить в представление сообщения об ошибках. Но на уровне *BLL* *ModelState* недоступен. Однако валидация с передачей ошибок в уровень представления всё же возможна.

Для этого в *BLL* определён класс *ValidationException*. Этот класс наследуется от базового класса исключений *Exception* и определяет свойство *Property*. Это свойство позволяет сохранить название свойства модели, которое некорректно и не проходит валидацию. И также передавая в конструктор базового класса параметр «*message*», определяется сообщение, которое будет выводиться для некорректного свойства в *Property*.

Взаимодействие между остальными уровнями будет происходить через специальный сервис. Для этого сервиса определён интерфейс *IWorkerService*. В качестве реализации этого интерфейса определён класс сервиса *WorkerService*. *WorkerService* в конструкторе принимает объект *IUnitOfWork*, через который идет взаимодействие с уровнем *DAL*.

Оновной функционал программы построен таким образом, что все модули *BLL*-слоя напрямую не взаимодействуют с хранилищем данных. Тем самым слои *BLL* и *DAL* наиболее отстранены друг от друга, что приводит к минимальным изменениям внутри компонентов при взаимодействии этих слоев. Итоговая структура слоя бизнес-логики представлена на рисунке 2.7.

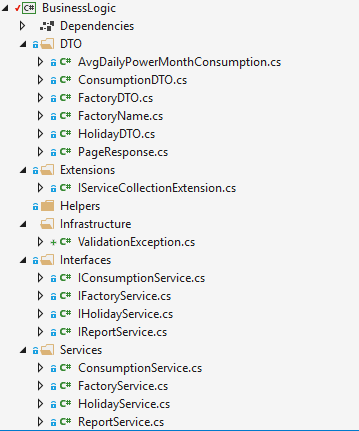


Рисунок 2.7 – Структура *BLL*-слоя

## 2.5 Слой представления (*Presentation Layer*)

Слой представления, или *Presentation* *Layer*, – это единственный слой в приложении с трехуровневой архитектурой, непосредственно связанный с пользователем. Он используется для получения данных от пользователя и передачи их *BLL*-слою для дальнейшей обработки, и представляет объект в надлежащей форме, которая понятна пользователю. Слой представления реализован как *web-*приложение *ASP*.*NET API* и приложение-клиент, написанное при использовании *JavaScript* фреймворка *Angular*.

***2.5.1*** Аббревиатура *API* расшифровывается как «*Application Programming Interface*» (интерфейс программирования приложений, программный интерфейс приложения) [6]. Большинство крупных компаний на определённом этапе разрабатывают *API* для клиентов или для внутреннего использования. Чтобы понять, как и каким образом *API* применяется в разработке и бизнесе, сначала нужно разобраться, как устроена «всемирная паутина».

«*WWW*» можно представить как огромную сеть связанных серверов, на которых и хранится каждая страница. Обычный ноутбук можно превратить в сервер, способный обслуживать целый сайт в сети, а локальные серверы разработчики используют для создания сайтов перед тем, как открыть их для широкого круга пользователей.

При введении в адресную строку браузера *www.facebook.com* на удалённый сервер *Facebook* отправляется соответствующий запрос. Как только браузер получает ответ, то интерпретирует код и отображает страницу.

Каждый раз, когда пользователь посещает какую-либо страницу в сети, он взаимодействует с *API* удалённого сервера. *API* – это составляющая часть сервера, которая получает запросы и отправляет ответы.

Слово «*application*» (прикладной, приложение) может применяться в разных значениях [7]. В контексте *API* оно подразумевает:

* фрагмент программного обеспечения с определённой функцией;
* сервер целиком, приложение целиком или же просто отдельную часть приложения.

Любой фрагмент ПО, который можно чётко выделить из окружения, может заменять букву «*А*» в англоязычной аббревиатуре, и тоже может иметь некоторого рода *API*. Например, при внедрении в код разработчиком сторонней библиотеки, она становится частью всего приложения. Будучи самостоятельным фрагментом ПО, библиотека будет иметь некий *API*, который позволит ей взаимодействовать с остальным кодом приложения.

В объектно-ориентированном проектировании код представлен в виде совокупности объектов. В приложении таких объектов, взаимодействующих между собой, могут быть сотни. У каждого из них есть свой *API* ­ набор публичных свойств и методов для взаимодействия с другими объектами в приложении. Объекты могут также иметь частную, внутреннюю логику, которая скрыта от окружения и не является *API*.

Для решения поставленной задачи был разработан *API*, содержащий 4 контроллера:

* *ConsumptionsController*;
* *FactoriesController*;
* *HolidaysController*;
* *ReportsController*.

*ConsumptionsController* отвечает за добавление и удаление информации об энергопотреблении предприятия.

*FactoriesController* обслуживает запросы по добавлению и удалению компаний, данные о потреблении энергии которых будут храниться.

*HolidaysController* нужен для работы с данными о выходных выпавших на рабочий день недели.

*ReportsController* отвечает за выдачу статистической информации.

Структура получившегося приложения изображена на рисунке 2.8.

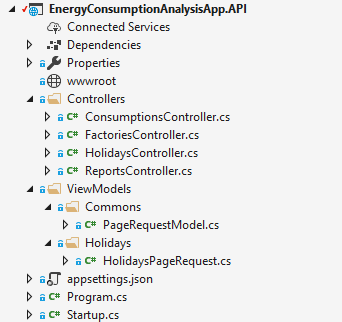


Рисунок 2.8 – Структура *API* приложения

Таким образом был разработан *API*, который полностью реализует функционал, необходимый для решения поставленной задачи.

***2.5.2*** *Angular* представляет фреймворк от компании *Google* для создания клиентских приложений. Прежде всего он нацелен на разработку *Single Page Application*, то есть одностраничных приложений. В этом плане *Angular* является наследником другого фреймворка *AngularJS*. В то же время *Angular* это не новая версия *AngularJS*, а принципиально новый фреймворк.

*Angular* предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизацию и т.д.

Преимущества *Angular*:

* *Angular* представляет не только инструменты, но и шаблоны дизайна для создания обслуживаемого проекта, при правильном создании *Angular* приложения не будет путаницы классов и методов, которые сложно править и еще сложнее тестировать, код удобно структурирован, можно быстро понять, что к чему;
* *Angular* построен на *TypeScript*, который, в свою очередь, полагается на *ES6*, т.е. нет необходимости учить полностью новый язык, и в тоже время предоставляются такие функции как статическая типизация, интерфейсы, классы, пространства имен, декораторы и другие;
* в *Angular* уже есть много инструментов для создания приложения, благодаря директивам;
* *HTML* элементы могут вести себя динамически;
* *FormControl* позволяет применить различные правила валидации;
* есть возможность легко посылать асинхронные *HTTP* запросы различных типов;
* встроенный механизм маршрутизации;
* компоненты разъединены: *Angular* старался убрать жесткую связь между различными компонентами приложения, внедрение проходит подобно *NodeJS*, что позволяет легко заменять компоненты.

Получившееся приложение-клиент на *Angular* имеет 7 страниц и реализует весь указанный в задании функционал.

# ЗАКЛЮЧЕНИЕ

Разработанное программное обеспечение позволяет автоматизировать работу по ведению учёта энергопотребления на различных предприятиях, а также автоматизирует процесс выборки и визуализации статистических данных.

В ходе технологической практики были решены следующие задачи:

* знания о подходах разработки приложений закреплены на практике;
* получен опыт работы на предприятии;
* получены навыки работы с командой;
* изучены и проанализированы предметная область, технологии и методы, необходимые для решения поставленных задач;
* разработаны алгоритмы решения поставленной задачи;
* на основе разработанных алгоритмов создано и протестировано программное обеспечение, решающее проблему ведения учёта энергопотребления на различных предприятиях, а также автоматизирующее процесс выборки и визуализации статистических данных.

# Список использованных источников

1. Шаблоны корпоративных приложений / Под ред. М. Фаулер. – Киев: Вильямс, 2016. – 38 с.
2. Комлев, Н. Ю. Объектно Ориентированное Программирование. Хорошая книга для Хороших Людей / Н. Ю. Комлев. – Москва: Дипак, 2014. – 23 с.
3. Фримен, Э. Паттерны проектирования / Э. Фримен, Э. Фримен. – СПб.: Питер, 2011. – 176 с.
4. Фримен, Э. Паттерны проектирования / Э. Фримен, Э. Фримен. – СПб.: Питер, 2011. – 153 с.
5. Фримен, Э. Паттерны проектирования / Э. Фримен, Э. Фримен. – СПб.: Питер, 2011. – 200 с.
6. Маклафлин, Б. Объектно-ориентированный анализ и проектирование / Б. Маклафлин, Г. Поллайс, Д. Уэст. – СПб.: Питер, 2013. – 231 с.
7. Рихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Д. Рихтер. – СПб.Ж Питер, 2017. – 17 с.

# ПРИЛОЖЕНИЕ А

(обязательное)

**Текст программы**

Класс *ConsumptionsController*:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using BusinessLogic.DTO;

using BusinessLogic.Interfaces;

using EnergyConsumptionAnalysisApp.API.ViewModels.Commons;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace EnergyConsumptionAnalysisApp.API.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class ConsumptionsController : ControllerBase

{

IConsumptionService consumptionService;

public ConsumptionsController(IConsumptionService consumptionService)

{

this.consumptionService = consumptionService;

}

[HttpPost]

[Route("Add")]

public IActionResult Add(ConsumptionDTO consumption)

{

consumptionService.AddConsumption(consumption);

return NoContent();

}

[HttpGet]

[Route("GetConsumptions")]

public IActionResult GetConsumptions(int factoryId, int? page, int? count)

{

var consumptions = consumptionService.GetConsumptions(factoryId, count ?? 25, page ?? 1);

return Ok(consumptions);

}

[HttpPost]

[Route("Delete/{id}")]

public IActionResult Delete(int id)

{

consumptionService.DeleteConsumption(id);

return NoContent();

}

}

}

Класс *FactoriesController* :

using BusinessLogic.DTO;

using BusinessLogic.Interfaces;

using Microsoft.AspNetCore.Mvc;

namespace EnergyConsumptionAnalysisApp.API.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class FactoriesController : ControllerBase

{

IFactoryService factoryService;

public FactoriesController(IFactoryService factoryService)

{

this.factoryService = factoryService;

}

[HttpPost]

[Route("Add")]

public IActionResult Add(FactoryDTO factory)

{

factoryService.AddFactory(factory);

return NoContent();

}

[HttpGet]

[Route("GetFactories")]

public IActionResult GetFactories(int? page, int? count)

{

var factories = factoryService.GetFactories(count ?? 25, page ?? 1);

return Ok(factories);

}

[HttpGet]

[Route("GetFactoryNames")]

public IActionResult GetFactoryNames()

{

var names = factoryService.GetFactoryNames();

return Ok(names);

}

[HttpPost]

[Route("Delete/{id}")]

public IActionResult Delete(int id)

{

factoryService.DeleteFactory(id);

return NoContent();

}

[HttpPost]

[Route("Update")]

public IActionResult Update(FactoryDTO factory)

{

factoryService.UpdateFactory(factory);

return NoContent();

}

}

Класс *HolidaysController*:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using BusinessLogic.DTO;

using BusinessLogic.Interfaces;

using EnergyConsumptionAnalysisApp.API.ViewModels.Commons;

using EnergyConsumptionAnalysisApp.API.ViewModels.Holidays;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace EnergyConsumptionAnalysisApp.API.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class HolidaysController : ControllerBase

{

IHolidayService holidayService;

public HolidaysController(IHolidayService holidayService)

{

this.holidayService = holidayService;

}

[HttpPost]

[Route("Add")]

public IActionResult Add(HolidayDTO holiday)

{

holidayService.AddHoliday(holiday);

return NoContent();

}

[HttpPost]

[Route("Delete/{id}")]

public IActionResult Delete(int id)

{

holidayService.DeleteHoliday(id);

return NoContent();

}

[HttpGet]

[Route("GetHolidays")]

public IActionResult GetHolidays(int factoryId, int? page, int? count)

{

PageResponse<HolidayDTO> holidays = holidayService.GetHolidays(factoryId, count ?? 25, page ?? 1);

return Ok(holidays);

}

//[HttpGet]

//[Route("GetHolidays")]

//public IActionResult GetHolidays(int factoryId, int year)

//{

// PageResponse<HolidayDTO> holidays = holidayService.GetHolidays(factoryId, year);

// return Ok(holidays);

//}

}

}

Интерфейс *IRepository.cs*:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HumanResourcesDepartment.DAL.Interfaces

{

public interface IRepository<T> where T : class

{

IEnumerable<T> GetAll();

T Get(int id);

IEnumerable<T> Find(Func<T, Boolean> predicate);

void Create(T item);

void Update(T item);

void Delete(int id);

}

}

Интерфейс *IUnitOfWork.cs*:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using HumanResourcesDepartment.DAL.Entities;

namespace HumanResourcesDepartment.DAL.Interfaces

{

public interface IUnitOfWork : IDisposable

{

IRepository<Worker> Workers { get; }

IRepository<WorkRecord> WorkRecords { get; }

void Save();

}

}

Класс *EfUnitOfWork.cs*:

using DataAccess.EF;

using DataAccess.Interfaces;

using DataAccess.Models;

using Microsoft.EntityFrameworkCore;

using System;

namespace DataAccess.Repositories.EFRepositories

{

public class EfUnitOfWork : IUnitOfWork

{

private ConsumptionsContext db;

private FactoriesEfRepository factoriesRepository;

private HolidaysEfRepository holidaysRepository;

private ConsumptionsEfRepository consumptionsRepository;

public EfUnitOfWork(string connStr)

{

var optionsBuilder = new DbContextOptionsBuilder<ConsumptionsContext>();

optionsBuilder.UseSqlServer(connStr);

db = new ConsumptionsContext(optionsBuilder.Options);

}

public IRepository<Factory> Factories

{

get

{

if (factoriesRepository == null)

{

factoriesRepository = new FactoriesEfRepository(db);

}

return factoriesRepository;

}

}

public IRepository<Holiday> Holidays

{

get

{

if (holidaysRepository == null)

{

holidaysRepository = new HolidaysEfRepository(db);

}

return holidaysRepository;

}

}

public IRepository<Consumption> Consumptions

{

get

{

if (consumptionsRepository == null)

{

consumptionsRepository = new ConsumptionsEfRepository(db);

}

return consumptionsRepository;

}

}

public void Save()

{

db.SaveChanges();

}

private bool disposed = false;

public virtual void Dispose(bool disposing)

{

if (!disposed)

{

if (disposing)

{

db.Dispose();

}

disposed = true;

}

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

}

}

Класс *ConsumptionsEfRepository*:

using System;

using System.Collections.Generic;

using System.Linq;

using DataAccess.EF;

using DataAccess.Interfaces;

using DataAccess.Models;

using Microsoft.EntityFrameworkCore;

namespace DataAccess.Repositories.EFRepositories

{

public class ConsumptionsEfRepository : IRepository<Consumption>

{

private ConsumptionsContext db;

public ConsumptionsEfRepository(ConsumptionsContext context)

{

db = context;

}

public void Create(Consumption obj)

{

db.Consumptions.Add(obj);

}

public void Delete(int id)

{

var item = db.Consumptions.Find(id);

if (item != null)

{

db.Remove(item);

}

}

public Consumption Get(int id)

{

return db.Consumptions.Find(id);

}

public IEnumerable<Consumption> GetAll()

{

return db.Consumptions;

}

public void Update(Consumption obj)

{

db.Entry(obj).State = EntityState.Modified;

}

public IEnumerable<Consumption> Where(Func<Consumption, bool> predicate)

{

return db.Consumptions.Where(predicate);

}

}

}

Класс *FactoriesEfRepository*:

using DataAccess.EF;

using DataAccess.Interfaces;

using DataAccess.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

namespace DataAccess.Repositories.EFRepositories

{

public class FactoriesEfRepository : IRepository<Factory>

{

private ConsumptionsContext db;

public FactoriesEfRepository(ConsumptionsContext context)

{

db = context;

}

public void Create(Factory obj)

{

db.Factories.Add(obj);

}

public void Delete(int id)

{

var item = db.Factories.Find(id);

if (item != null)

{

db.Remove(item);

}

}

public Factory Get(int id)

{

return db.Factories.Find(id);

}

public IEnumerable<Factory> GetAll()

{

return db.Factories;

}

public void Update(Factory obj)

{

db.Entry(obj).State = EntityState.Modified;

}

public IEnumerable<Factory> Where(Func<Factory, bool> predicate)

{

return db.Factories.Where(predicate);

}

}

}

Класс *ConsumptionDTO*:

using System;

namespace BusinessLogic.DTO

{

public class ConsumptionDTO

{

public int Id { get; set; }

public int FactoryId { get; set; }

public DateTime Date { get; set; }

public double Value { get; set; }

}

}

Класс *FactoryDTO.cs*:

namespace BusinessLogic.DTO

{

public class FactoryDTO

{

public int Id { get; set; }

public string Name { get; set; }

public string Units { get; set; }

}

}

Интерфейс *IConsumptionService.cs*:

using BusinessLogic.DTO;

using System.Collections.Generic;

namespace BusinessLogic.Interfaces

{

public interface IConsumptionService

{

PageResponse<ConsumptionDTO> GetConsumptions(int factoryId, int count, int page);

void AddConsumption(ConsumptionDTO consumption);

void DeleteConsumption(int id);

void Dispose();

}

}

Интерфейс *IFactoryService*:

using BusinessLogic.DTO;

using System.Collections.Generic;

namespace BusinessLogic.Interfaces

{

public interface IFactoryService

{

PageResponse<FactoryDTO> GetFactories(int count, int page);

void AddFactory(FactoryDTO factory);

void DeleteFactory(int id);

void UpdateFactory(FactoryDTO factory);

IEnumerable<FactoryName> GetFactoryNames();

void Dispose();

}

}

Класс *ConsumptionService.cs*:

using BusinessLogic.DTO;

using BusinessLogic.Interfaces;

using DataAccess.Interfaces;

using DataAccess.Models;

using System.Linq;

namespace BusinessLogic.Services

{

public class ConsumptionService : IConsumptionService

{

IUnitOfWork db;

public ConsumptionService(IUnitOfWork unit)

{

db = unit;

}

public void AddConsumption(ConsumptionDTO consumption)

{

var cons = new Consumption

{

Date = consumption.Date,

FactoryId = consumption.FactoryId,

Value = consumption.Value

};

db.Consumptions.Create(cons);

db.Save();

}

public void DeleteConsumption(int id)

{

db.Consumptions.Delete(id);

db.Save();

}

public PageResponse<ConsumptionDTO> GetConsumptions(int factoryId, int count, int page)

{

var skip = count \* (page - 1);

var consumptions = db.Consumptions.Where(cons => cons.FactoryId == factoryId);

var result = consumptions.OrderByDescending(cons => cons.Date)

.Skip(skip)

.Take(count)

.Select(cons => new ConsumptionDTO

{

Date = cons.Date,

FactoryId = cons.FactoryId,

Id = cons.Id,

Value = cons.Value

});

return new PageResponse<ConsumptionDTO> { Items = result, Total = consumptions.Count() };

}

public void Dispose()

{

db.Dispose();

}

}

}

Класс *FactoryService.cs*:

using BusinessLogic.DTO;

using BusinessLogic.Interfaces;

using DataAccess.Interfaces;

using DataAccess.Models;

using System.Collections.Generic;

using System.Linq;

namespace BusinessLogic.Services

{

public class FactoryService : IFactoryService

{

IUnitOfWork db;

public FactoryService(IUnitOfWork unit)

{

db = unit;

}

public void AddFactory(FactoryDTO factory)

{

db.Factories.Create(new Factory

{

Name = factory.Name,

Units = factory.Units

});

db.Save();

}

public void DeleteFactory(int id)

{

db.Factories.Delete(id);

db.Save();

}

public PageResponse<FactoryDTO> GetFactories(int count, int page)

{

var skip = count \* (page - 1);

var factories = db.Factories.GetAll();

var result = factories.OrderBy(fact => fact.Name).Skip(skip).Take(count).Select(fact => new FactoryDTO

{

Id = fact.Id,

Name = fact.Name,

Units = fact.Units

});

return new PageResponse<FactoryDTO> { Items = result, Total = factories.Count() };

}

public IEnumerable<FactoryName> GetFactoryNames()

{

var factories = db.Factories.GetAll().Select(fact => new FactoryName

{

Id = fact.Id,

Name = fact.Name

});

return factories;

}

public void UpdateFactory(FactoryDTO factory)

{

var fact = db.Factories.Get(factory.Id);

fact.Name = factory.Name;

fact.Units = factory.Units;

db.Factories.Update(fact);

db.Save();

}

public void Dispose()

{

db.Dispose();

}

}

}

Класс *HolidaysController.cs*:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using BusinessLogic.DTO;

using BusinessLogic.Interfaces;

using EnergyConsumptionAnalysisApp.API.ViewModels.Commons;

using EnergyConsumptionAnalysisApp.API.ViewModels.Holidays;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace EnergyConsumptionAnalysisApp.API.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class HolidaysController : ControllerBase

{

IHolidayService holidayService;

public HolidaysController(IHolidayService holidayService)

{

this.holidayService = holidayService;

}

[HttpPost]

[Route("Add")]

public IActionResult Add(HolidayDTO holiday)

{

holidayService.AddHoliday(holiday);

return NoContent();

}

[HttpPost]

[Route("Delete/{id}")]

public IActionResult Delete(int id)

{

holidayService.DeleteHoliday(id);

return NoContent();

}

[HttpGet]

[Route("GetHolidays")]

public IActionResult GetHolidays(int factoryId, int? page, int? count)

{

PageResponse<HolidayDTO> holidays = holidayService.GetHolidays(factoryId, count ?? 25, page ?? 1);

return Ok(holidays);

}

//[HttpGet]

//[Route("GetHolidays")]

//public IActionResult GetHolidays(int factoryId, int year)

//{

// PageResponse<HolidayDTO> holidays = holidayService.GetHolidays(factoryId, year);

// return Ok(holidays);

//}

}

}

Класс *WorkerViewModel.cs*:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.ComponentModel.DataAnnotations;

namespace HumanResourcesDepartment.WEB.Models

{

public class WorkerViewModel

{

public int Id { get; set; }

[Required(ErrorMessage = "Обязательное для ввода поле")]

[RegularExpression(@"([A-Z][a-z]+)|([А-Я][а-я]+)", ErrorMessage = "Имя должно состоять только из букв, начинаться с большой буквы " +

"и состоять минимум из двух букв")]

public string FirstName { get; set; }

[RegularExpression(@"([A-Z][a-z]+)|([А-Я][а-я]+)", ErrorMessage = "Фамилия должно состоять только из букв, начинаться с большой буквы " +

"и состоять минимум из двух букв")]

[Required(ErrorMessage = "Обязательное для ввода поле")]

public string SecondName { get; set; }

[RegularExpression(@"([A-Z][a-z]+)|([А-Я][а-я]+)", ErrorMessage = "Отчество должно состоять только из букв, начинаться с большой буквы " +

"и состоять минимум из двух букв")]

[Required(ErrorMessage = "Обязательное для ввода поле")]

public string NativeName { get; set; }

[Required(ErrorMessage = "Обязательное для ввода поле")]

[DataType(DataType.Date)]

public DateTime BirthDate { get; set; }

}

}